HCI and Software Engineering for User Interface Plasticity

Joëlle Coutaz, Gaëlle Calvary

52

HCI AND SOFTWARE ENGINEERING FOR USER INTERFACE PLASTICITY

Joëlle Coutaz and Gaëlle Calvary
University of Grenoble, Laboratory of Informatics of Grenoble (LIG)

published as Chapter 52 in "The Human-Computer Handbook – Fundamentals, Evolving Technologies, and Emerging Applications", 3rd edition, Julie, A. Jacko ed., CRC Press Taylor and Francis Group, p. 1195-1220, 2012

1. INTRODUCTION

Human-computer interaction (HCI) and Software Engineering (SE) are like two old friends with different backgrounds: they share values but use them differently. Both domains address the design and development of useful and usable systems and are concerned with "requirements analysis," "incremental and iterative design," as well as "quality assurance." However, they address these problems with different development processes, different notations, and different priorities. For HCI, the human is the first-class entity in all phases of development. For SE, the final objective is a running system developed at minimal cost and delivered in time, while satisfying contractual specifications. The user is, at best, involved at the very beginning of the process, and hopefully at the very end of the project for summative evaluation. However, to avoid or correct wrong design decisions, this is too little and too late. Even in the early stages of development, functional requirements and quality goals are rarely the result of a close collaboration between HCI and SE specialists.

There are many reasons for the lack of collaboration between HCI and SE scientists and practitioners: mutual ignorance resulting from educational background, and from there, economic consideration. HCI methods such as contextual design (see chapter 49—Karen Holtzblatt), scenario-based approaches (Rosson et al., 2002), and task analysis are perceived as too demanding in terms of time and competence to inform system requirements in a formal and timely manner. On the other hand, Unified Modeling Language (UML) use cases, which express the functions that the system should support with a scenario-based flavor, are pale attempts to factor out user-centered concerns. They do not result from a human-centered requirements analysis nor do they have the expressive power of task models. Task-modeling techniques, such as Concur Task Tree (CTT) (Paternò, 2003) or User Action Notation (UAN) (Hartson, 1990), which use notations familiar to computer scientists (i.e. (LOTOS) Language of Temporal Ordering Specification operators and logic), are not used in software engineering. On the other hand, task models are not well suited for expressing what can go wrong whereas the software engineering KAOS goal-oriented modeling approach supports the explicit expression of "goal obstacles" (van Lamsweerde, 2009). Similarly, domain-dependent concepts referenced in task models are ill defined, whereas UML class diagrams would improve task specifications significantly.

In summary, HCI and SE pursue the same goal, using development processes and notations that sometimes overlap and complement each other. In this chapter, we present one way to exploit both fields for the development of plastic user interfaces using the notion of model as the keystone between the two disciplines. In the following sections, we define the concept of user interface (UI) plasticity and develop the problem space for this concept. Exemplars of plastic interactive systems will illustrate aspects of this problem space. We then introduce the key objectives and principles of Model-Driven Engineering (MDE) (http://planetmde.org) and analyze the contributions and limitations of MDE to address the problem of UI plasticity. Drawing from our experience with MDE applied to UI plasticity, we show how to address these limitations and conclude with recommendations for a research agenda.

2. USER INTERFACE PLASTICITY: DEFINITION

The term plasticity is inspired from the capacity of biological tissues such as plants and brain, to undergo continuous deformation to rebuild themselves and to adapt to external constraints to preserve function without rupture. Applied to interactive systems, *UI plasticity is the capacity of user interfaces to adapt to the context of use while preserving usability* (Thevenin & Coutaz, 1999) *or human values* (Calvary et al., 2003; Cockton, 2004)[1]. In the following sections, we define context of use, usability as well as the notion of human values in more detail.

2.1 Context and Context of Use

Since the early sixties, the notion of context has been modeled and exploited in many areas of informatics. The scientific community has debated definitions and uses for many years without reaching clear consensus (Dourish, 2001; Dey 2001). Nonetheless, it is commonly agreed that context is about evolving, structured, and shared information spaces (Winograd, 2001), and that such spaces are designed to serve a particular purpose (Coutaz et al., 2005). In UI plasticity, the purpose is to support the adaptation process of the user interface to preserve use. Thus, there is no such thing as "the" context, but there is a context qualified by the process it serves. This is why we use the term "context of use", and not simply the word context. A context change could be defined as the modification of the value of any element of the contextual information spaces. This definition would lead to an explosion of contexts. The following ontological foundation provides some structure to master this explosion.

2.1.1 Ontological Foundation for Context

As shown in Fig. 52.1, a contextual information space is modeled as a directed graph where a node denotes a context and an edge denotes a condition to move between two contexts. In turn, a context is a directed graph of situations where a node denotes a situation and an edge denotes a condition to move between two situations. Thus, a contextual information space is a two-level data structure, i.e. a graph of contexts where each context is in turn a graph of situations. If more structure is needed, situations may in turn be refined into "sub-situations" and so on. We now need to specify the domain of definition of contexts and situations.

A context is defined over a set E of *entities*, a set Ro of *roles* (i.e. functions) that these entities may satisfy, and a set Rel of *relations* between entities. Entities, roles, and relations are modeled as expressions of *observables* that are captured and/or inferred by the system. For example, in a conference room, E denotes the participants, Ro denotes the roles of speaker and listener, and Rel denotes some spatial relations between entities such as "entity e1 (who plays the role of speaker) stands in front of entity e2 (who plays the role of a listener)". The situations that pertain to the same context share the same sets E, Ro, and Rel.

The condition to move between two contexts is one of the following: E is replaced by a different set (e.g., the set E of participants is now replaced with the set E' of family members), Ro has changed (e.g., the roles of speaker and listener are replaced with that of parent), or Rel has changed (e.g., in addition to spatial relationships, temporal relationships between entities may now matter).

The condition to move between two situations is one of the following:

- The cardinality of the set E has changed. For example, 2 persons enter the room and are recognized by the system as participants (their observables match the characteristics and behavior of participants). As a result, the system may provide the two latecomers with a summary of the current talk. If recognized as the organizers of the conference, then the system would detect a context change (not a situation change) because a new role (i.e. that of an organizer) is coming into play.
- A role assignment to an entity has changed (e.g., participant e switches from speaker to listener),
- A relation between two entities has changed (e.g., participant e was in front of e'. Now, e' is in front of e).

[1] The cloud computing community uses the term "elasticity" for systems where "capabilities can be rapidly and elastically provisioned, in some cases automatically, to quickly scale out and rapidly released to scale in. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be purchased in any quantity at any time" (NIST's definition – http://csrc.nist.gov/groups/SNS/cloud-computing/). Indeed, elasticity has the property of returning to an initial form (or state) following strain, which is not necessarily the case for plasticity. An elastic matter can break whereas a plastic entity aims at preserving survival. For these reasons, we consider that plasticity is a more general and demanding property than elasticity. We must admit however, that they are very similar in spirit. (In economics, elasticity measures the incidence of the variation of one variable on that of another variable. Cf. Wikipedia.)

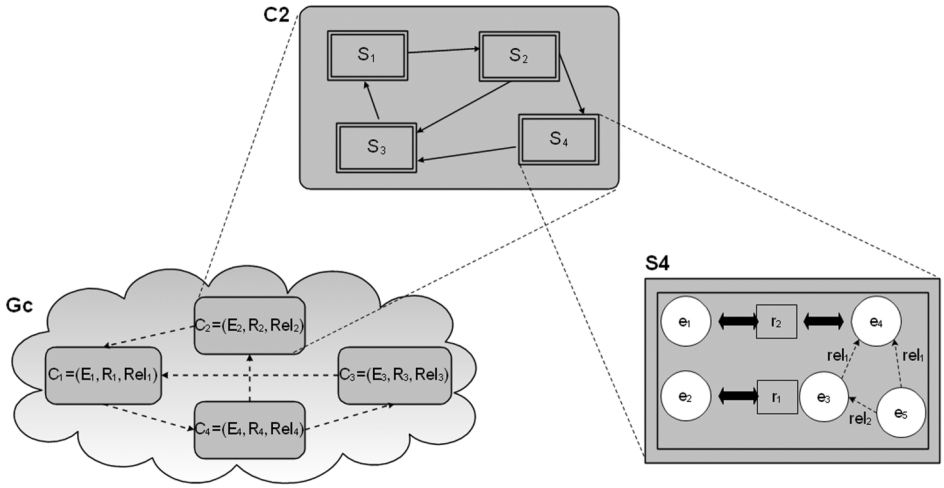


Figure 52.1. The graph of contexts Gc is composed of 4 contexts C1, C2, C3, C4 defined on their own sets of Entities, Roles, and Relations. In turn, Context C2 is composed of 4 situations S1, S2, S3, S4. By definition, these situations share the same sets of Entities, Roles, and Relations. In S4, Entities e1 and e4 (elements of E4) play the role r2 (element of R4), whereas Role r1 is played by Entities e2 and e3; e3 and e4 satisfy Relation rel1, e5 and e3 satisfy rel2, and e5 and e4 are related by rel1.

The ontology does not specify the nature of the entities, roles, relations, and observables. These are abstract classes from which a domain-dependent model can be specified. Using expressions of observables, designers identify the set of entities, roles, and relations that are relevant for the case at hand. Reignier et al. use this ontology and these principles for moving between situations and contexts for the development of smart environments (Reignier et al., 2007). For the purpose of UI plasticity, the observables of a context of use are organized into three information spaces that model the user, the environment, and the computing platform, respectively.

2.1.2 Observables of the Context of Use

The observables of a context of use define three information spaces respectively called (a) the user model, (b) the environment model, and (c) the platform model.

- The user model denotes the attributes and functions that describe the archetypal person who is intended to use, or is actually using, the interactive system. This ranges from basic user preferences as provided by most interactive systems, to more sophisticated descriptions such as profiles, idiosyncrasies and current activities inferred from the repetitive use of services, of commands sequences and current tasks.
- The environment model includes attributes and functions that characterize the physical places and times where the interaction will take place or is actually taking place. As for the user model, the number of candidate dimensions is quite large. It includes numeric locations (e.g., GPS coordinates) and/or symbolic locations (e.g., at home, in a public space, on the move in the street, a train or a car), numeric and symbolic temporal characteristics (e.g., 4th of January VS winter), social rules and activities, as well as physical human perceivable conditions such as light, heat, and sound conditions (using numeric and/or symbolic representations).
- The platform model describes the computing, sensing, networking, and interaction resources that bind together the physical environment with the digital world. In the conventional GUI paradigm, the platform is limited to a single computing device, typically a workstation or a smart phone, connected to a network and equipped with a fixed set of interaction resources such as a screen, keyboard, and stylus. Technological advances are leading to the capacity for individuals to assemble and mould their own interactive spaces from public hot spots and private devices to access services within the global computing fabric. Interactive spaces will soon take the form of autonomous computing islands, or ecosystems, whose horizon will evolve, split, and merge under human control. Resources will be coupled opportunistically to amplify human activities where any real-world object has the potential to play the role of an interaction resource. Among many others, the Shiftables (Merrill, 2007), the History Tablecloth (Gaver, 2006) as well as SkinInput (Harrison, 2010) illustrate this trend. As a result, the platform must be modeled as a dynamic cluster of heterogeneous resources, rather than as a conventional mono-computing static device.

2.2 Usability

The term *usability* is interpreted in different ways by authors, even within the same scientific community. Usability has been identified with ease of use and learning, while excluding utility (Shackel, 1984; Nielsen, 1993). In other cases, usability is used to denote ease of use and utility, while ignoring learning. In software engineering, usability is considered an intrinsic property of the software product, whereas in HCI, usability is contextual: a system is not intrinsically usable or unusable. Instead, usability arises relatively to contexts of use.

The contextual nature of usability has been recently recognized by the International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) 9126 standards developed in the software community with the overarching notion of "quality in use." Quality in use is "the capability of the software product to enable specified users to achieve specified goals with effectiveness, productivity, safety, and satisfaction in specified contexts of use." Unfortunately, as shown in Fig. 52.2, usability is viewed as one independent contribution to quality in use. Thus, the temptation is high for software people to assimilate usability to cosmetic issues limited to the user-interface component of a software product, forgetting that system latency, reliability, missing functions, and inappropriate sequencing of functions, have a strong impact on the system "useworthiness."

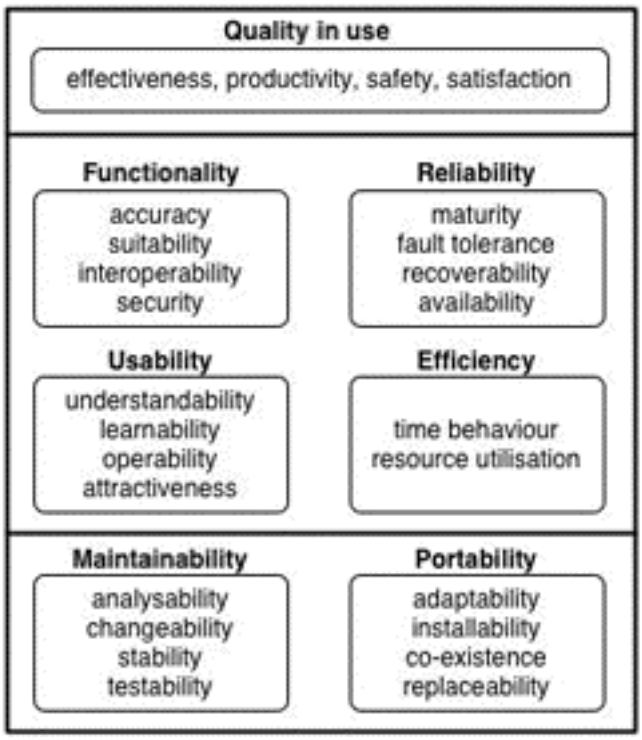


Figure 52.2 Usability model from ISO/IEC 9126-1.

Useworthiness is central to Cockton's argument for the development of systems that have value in the real world (Cockton 2004, 2005). In value-centered approaches, software design should start from the explicit expression of an intentional creation of value for a selected set of target contexts of use. Intended value for target contexts are then translated into evaluation criteria. Evaluation criteria are not necessarily elicited from generic intrinsic features such as time for task completion, but are contextualized. They are monitored and measured in real usage to assess the achieved value.

Building on Cockton's approach, we suppose that for each of the target contexts of use C_i of a system, an intended value V_i has been defined, and that V_i has been translated into the set of triples $\{(c_{i1}, d_{i1}, w_{i1}), \ldots, (c_{ij}, d_{ij}, w_{ij}), \ldots (c_{in}, d_{in}, w_{in})\}$ where c_{ij} is an evaluation criteria, and d_{ij} and w_{ij}, the expected domain of values and relative importance (the weight) of c_{ij} in C_i. As just discussed, c_{ij} may be a generic measurable feature or a customized measure that depends on the intended value in C_i. Usability U_i of the system for context C_i is evaluated against a combining function F_i on the set $\{(c_{i1}, d_{i1}, w_{i1}), \ldots, (c_{ij}, d_{ij}, w_{ij}), \ldots (c_{in}, d_{in}, w_{in})\}$ whose result is intended to lie within a domain of values D_i.

Coming back to the notion of plasticity, *an interactive system S is plastic from a source context of use C_i to a target of use C_j* if the following two conditions are satisfied:

1. Adaptation, if needed, is supported when switching from C_i to C_j
2. Usability (value) is preserved in C_j by the adaptation process. In other words, the usability function F_j defined for C_j lies within its intended domain D_j.

The domain of plasticity of a system is the set C of contexts of use C_i for which usability is achieved. We have defined usability by reasoning at the context level. If needed, a finer grain of reasoning can be applied at the situation level: intended value is defined for each situation of each context, and then translated into evaluation criteria. Preserving usability is then evaluated on situation changes.

These definitions provide a theoretical framework where value comes first and is defined on a per-context (or situation) of use basis. For each of the intended target contexts (or situations), value is operationalized into a mix of generic and customized metrics. The difficulty is the identification of the relevant contexts of use and situations as well as the appropriate translation of value into significant metrics. We have no answer for operationalizing value, except to use generic measures when applicable, to instrument the system appropriately using sound software development techniques, such as Aspect Oriented Programming (AOP) (Elrad, 2001), and to apply a healthy dose of common sense. On the other hand, our ontological framework on context and its associated method (Rey, 2005), can be used to define the boundaries of contexts and situations of use as well as their relationships. For our notion of context of use, the fundamental entities are the user(s), environment, and platform, each of them being characterized by observables monitored by the system. Section "Models at runtime" shows how to integrate the monitoring of

observables within the software architecture of an interactive system.

3. THE PROBLEM SPACE OF USER INTERFACE PLASTICITY

Figure 52.3 captures the problem space of UI plasticity where each branch denotes an issue along with the possible options for resolution. This problem space is characterized (but not limited to) the following dimensions: the means used for adaptation (i.e. remolding and redistribution); the smallest UI units that can be adapted by the way of these means (from the whole UI considered as a single piece of code to the finest grain: the interactor); the granularity of state recovery after adaptation has occurred (from the session level to the user's last action); the UI deployment (static or dynamic) as a way to characterize how much adaptation has been pre-defined at design-time VS computed at runtime; the context coverage to denote the causes for adaptation with which the system is able to cope; the coverage of the technological spaces as a way to characterize the degree of technical heterogeneity that the system supports; and the existence of a meta-UI to allow users to control and evaluate the adaptation process. A subset of these dimensions is now developed in detail.

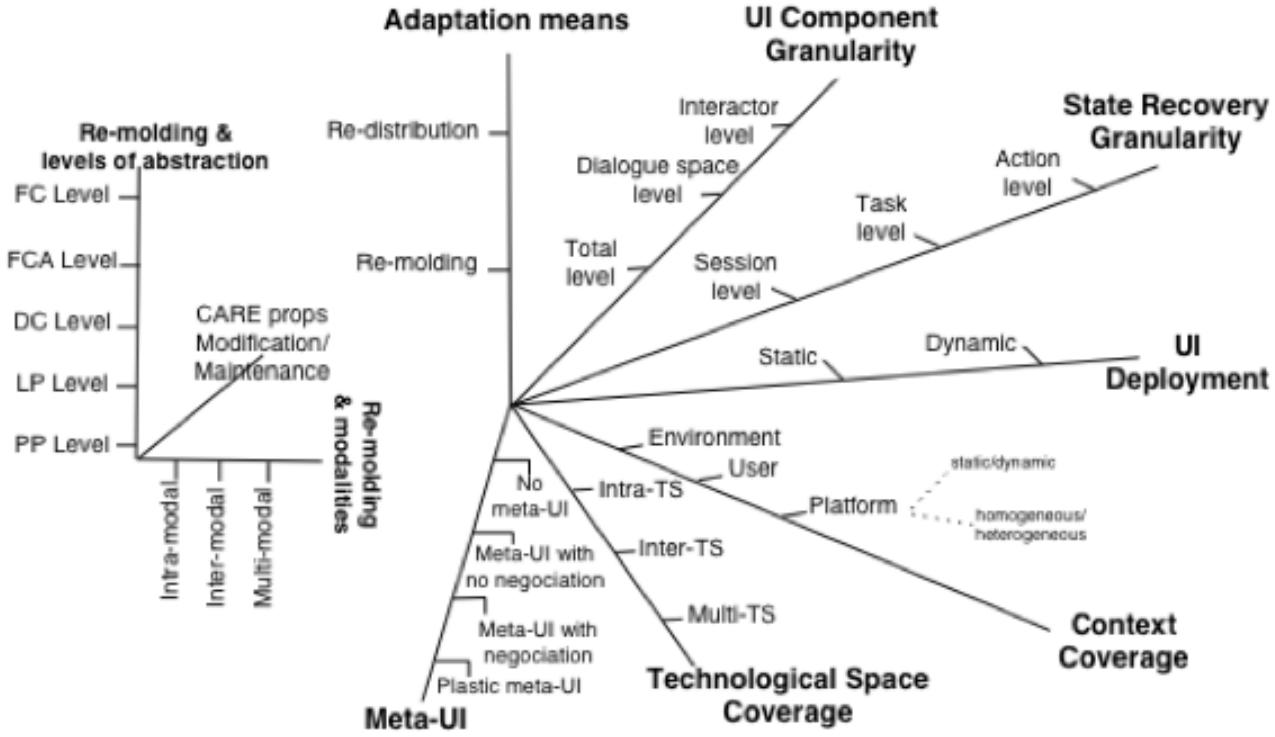


Figure 52.3 Problem space of user interface plasticity.

3.1 Adaptation means: UI remolding and UI redistribution

3.1.1 UI remolding

UI remolding consists in changing the "shape" of the user interface by applying one or several transformations on all, or parts, of the user interface. These transformations include: Suppression of the UI components that become irrelevant in the new situation/context; Insertion of new UI components to provide access to new services relevant in the new situation/context; Substitution of UI components when UI components are replaced with new ones. (Substitution can be viewed as a combination of suppression and insertion.); Reorganization of UI components by revisiting their spatial layout and/or their temporal dependency.

Remolding a user interface from a source to a target UI may imply changes in the set of the available modalities. UI remolding is *intra-modal* when the source UI components that need to be changed are retargeted within the same modality. Note that if the source user interface is multi-modal, then, the target UI is multi-modal as well: intra-modal remolding does not provoke any loss in the set of modalities. Remolding is *inter-modal* when the source UI components that need to be changed are retargeted into a different modality. Inter-modal retargeting may engender a modality loss or a modality gain. Thus, a source multi-modal UI may be retargeted into a mono-modal UI and conversely, a mono-modal UI may be transformed into a multi-modal

UI. Remolding is *multi-modal* when it uses a combination of intra- and inter-modal transformations. For example, TERESA supports multi-modal remolding between graphics and vocal modalities (Berti & Paternò, 2005).

Remolding a user interface may also change the way the CARE properties (Coutaz et al., 1995) are supported by the target UI (*Complementarity* – several modalities must be combined to produce a semantically valid expression whether it be for input or output; *Assignment* – only one modality can be used to produce a semantically valid input or output expression; *Redundancy* – several equivalent modalities are used simultaneously to produce a semantically valid input or output expression; *Equivalence* – several modalities are available to produce semantically equivalent expressions, but only one can be used at a time). Typically, because of a lack of computing power in the new situation, redundancy may be replaced by equivalence. Or, a synergistic-complementarity (as in the "put that there" vocal sentence produced in parallel with two deictic gestures to denote "that" and "there") may be transformed into an alternate-complementarity (as in the sequence: vocal "put that"; deictic gesture "that"; vocal "there" ; deictic gesture "there").

Transformations are performed at multiple levels of abstraction from cosmetic arrangements to deep software reconfiguration:

- At the Physical Presentation (PP) level, physical interactors (widgets) used for representing domain-dependent functions and concepts are kept unchanged but their rendering and behavior may change. For example, if a concept is rendered as a button class, this concept is still represented as a button in the target UI. However, the look and feel of the button or its location in the workspace may vary. This type of adaptation is used in Tk as well as in Java/AWT with the notion of peers.
- At the Logical Presentation (LP) level, adaptation consists of changing the representation of domain-dependent functions and concepts. For example, the concept of month can be rendered as a Label1Textfield, or as a Label1Combobox, or as a dedicated physical interactor. In an LP adaptation, physical interactors can replace each other provided that their representational and interactional capabilities are equivalent. The implementation of an LP-level adaptation can usefully rely on the distinction between Abstract Interactive Objects and Concrete Interactive Objects as presented in (Vanderdonckt & Bodart, 1993). Changes at the LP level imply changes at the PP level.
- At the Dialog Component (DC) level, the tasks that can be executed with the system are kept unchanged but their organization is modified. As a result, the structure of the dialogue structure is changed. AVANTI's polymorphic tasks (Stephanidis & Savidis, 2001) are an example of a DC-level adaptation. Changes at the DC level imply changes at the LP and PP levels.
- At the Functional Core Adaptor (FCA) level, the nature of the entities as well as the functions exported by the functional core (which implements the domain-dependent concepts and functions) are changed. Changes at the FCA level imply changes at the DC, LP, and PP levels.

UI adaptation is often assimilated to UI remolding. This is true as long as we live in a closed world where the interaction resources are limited to that of a single computer at a time. In ubiquitous computing, the platform may be a dynamic cluster composed of multiple interconnected computing devices. In this kind of situation, instead of being *centralized*, the user interface may be *distributed* across the interaction resources of the cluster.

3.1.2 UI redistribution

UI redistribution denotes the re-allocation of the UI components of the system to different interaction resources. The granularity of UI redistribution may vary from application level to pixel level:

- At the application level, the UI is fully replicated on each computing device. When the redistribution is dynamic, the whole UI of the application *migrates* to a new computing device, which in turn may trigger remolding.
- At the workspace level, the unit for distribution is the workspace. A workspace is a logical space that supports the execution of a set of logically connected tasks. This concept is similar to the notion of focus area used in contextual design for expressing the user-environment design. PebblesDraw (Myers, 2001) and Rekimoto's Pick and Drop (Rekimoto, 1997) are examples of UI distribution at the workspace level.
- The interactor level distribution is a special case of the workspace level where the unit for distribution is an elementary interactor.
- At the pixel level, any user interface component can be partitioned across multiple resources. For example, in the seminal smart room DynaWall (Streitz et al., 1999), a window may simultaneously lie over two contiguous white boards as if these were managed by a single computer.

3.2 State Recovery

The granularity of state recovery characterizes the effort users must apply to carry on their activity after adaptation has occurred. State recovery can be performed at the session, task, and physical action levels:

- When the system state is saved at the session level, users have to restart the interactive system from scratch. They rely on the state saved by the functional core before adaptation is taking place.
- At the task level, the user can pursue the job from the beginning of the current interrupted task (provided that the task is attainable in the retargeted system).
- At the physical action level, the user is able to carry on the current task at the exact point within the current task (provided that the task is attainable in the retargeted system).

3.3 Coverage of Technological Spaces

"A technological space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities." (Kurtev, 2002). Examples of technological spaces include documentware concerned with digital documents expressed in XML, dataware related to data base systems, ontologyware, and so on. Most user interfaces are implemented within a single Technological Space (TS), such as Tcl/Tk, Swing, html. This homogeneity does not hold anymore for plastic UI's since redistribution to different computing devices may require crossing technological spaces. For example, a Java-based UI must be transformed into WML when migrating from a PDA to a WAP-enabled mobile phone.

TS coverage denotes the capacity of the underlying infrastructure to support UI plasticity across technological spaces: *Intra-TS* corresponds to UIs that are implemented and adapted within a single TS. *Inter-TS* corresponds to the situation where the source UI, which is expressed in a single TS, is transformed into a single distinct target TS. *Multi-TS* is the flexible situation where the source and/or the target user interfaces are expressed in distinct technological spaces as supported by the Comet toolkit (Demeure et al., 2008).

3.4 Existence of a Meta-UI (or Supra-UI)

A meta-UI (or a Supra-UI) is a special kind of end-user development environment whose set of functions is necessary and sufficient to control and evaluate the state of an interactive ambient space (Coutaz, 2006). This set is *meta-* (or *supra-*) because it serves as an umbrella *beyond* the domain-dependent services that support human activities in this space. It is *UI-*oriented because its role is to allow users to control and evaluate the state of the ambient interactive space. By analogy, a meta-UI is to ambient computing what desktops and shells are to conventional workstations.

A *meta-UI without negotiation* makes observable the state of the adaptation process, but does not allow the user to intervene. A *meta-UI incorporates negotiation* when, for example, it cannot make sound decisions between multiple forms of adaptation, or when the user must fully control the outcome of the process.

The balance between system autonomy and too many negotiation steps is an open question. Another issue is the *plasticity of the meta-UI* itself. Thus, the recursive dimension of the meta-UI calls for the definition of a *native bootstrap meta-UI* capable of instantiating the appropriate meta-UI as the system is launched. This is yet another research issue.

The examples presented next illustrate the problem space of plastic UI's.

4. CASE STUDIES

CamNote and Sedan-Bouillon are two examples of plastic interactive systems developed according to the MDE approach presented next. The services they provide are accessible from different types of computing devices including workstations, personal digital assistants (PDA), and mobile phones. The UI components of these systems can be dynamically distributed and migrated across the interaction resources currently available in the interactive space. CamNote and Sedan-Bouillon differ in the technological spaces used for implementation: CamNote is Java-centric whereas Sedan-Bouillon uses PHP-MySQL Internet solutions. Whereas CamNote and Sedan-Bouillon offer a WIMP user interface, the user interface of our third example, Photo-Browser, includes a post-WIMP UI component. Photo-Browser has been developed to show how runtime adaptation can combine MDE with a code-centric approach.

4.1 CamNote

CamNote (for CAMELEON Note) is a slides viewer that runs on a dynamic heterogeneous platform. This platform may range from a single PC to a cluster composed of a PC and of a PDA. Its UI is structured into four workspaces: (a) a slides viewer, (b) a note editor for associating comments to the slides, (c) a video viewer also known as "mirror pixels" that shows a live video of the speaker, and (d) a control panel to browse the slides and to setup the level of transparency of the mirror. Speakers can point at items on the slide using their finger. This means of pointing is far more compelling and engaging than the conventional mouse pointer that no one can see. (Technically, the mirror is combined with the slides viewer using alpha-blending. See http://iihm.imag.fr/demos/CamNote/camnote_short.mov for a short movie demo.)

Figure 52.4a shows a configuration where the graphical UI is distributed across the screens of a PC and of a PDA. The slides viewer is displayed in a rotative canvas so that it can be oriented appropriately when projected onto a horizontal surface. If the PDA disappears, the control panel automatically migrates to the PC screen. Because different resources are now available, the control panel includes different widgets, but also a miniature representation of the speaker's video is now available. During the adaptation process, users can see the control panel emerging progressively from the slides viewer so that they can evaluate the progress of the adaptation process. The UI, which was distributed on a PC and a PDA, is now centralized on the PC (Fig. 52.4b). Conversely, if the PDA reenters the interactive space, the UI automatically switches to the configuration of Fig. 52.4a, and the control panel disappears from the PC screen by weaving itself into the slides viewer before reappearing on the PDA.

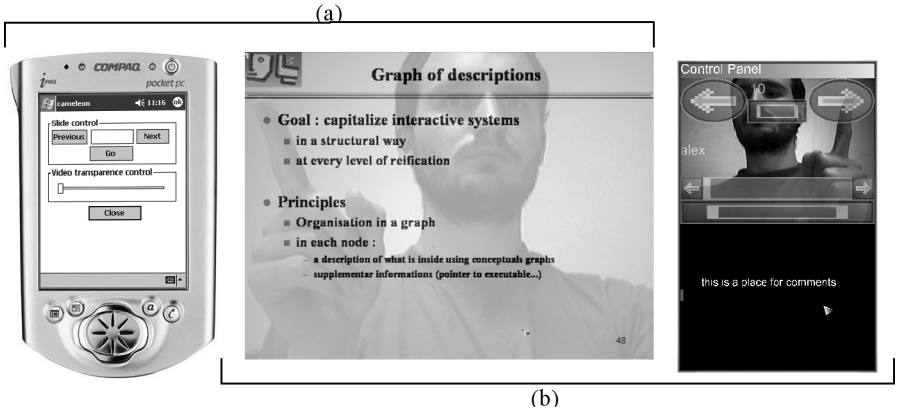
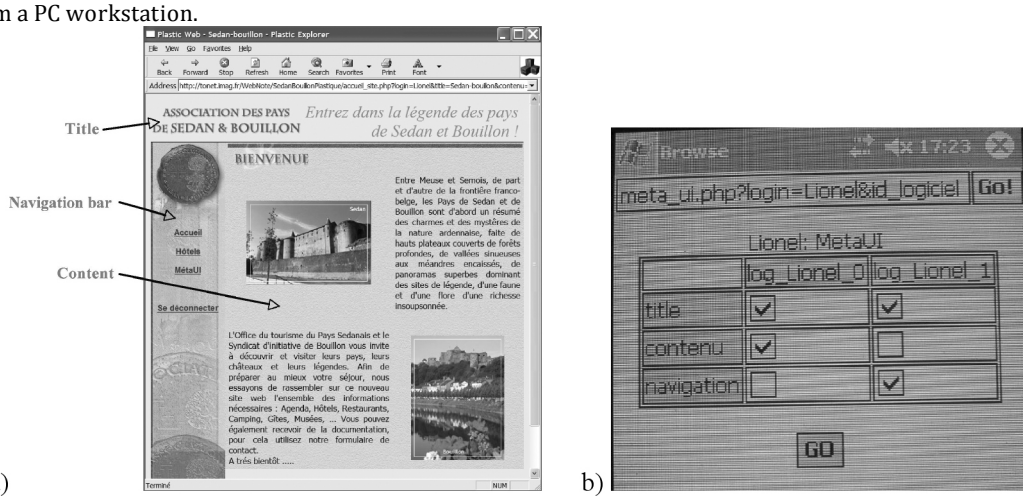


Figure 52.4 The user interface of CamNote. (a) The user interface of CamNote when distributed on a PC and a PocketPC screens; (b) the control panel when displayed on the PC screen.

In this exemplar, context of use is limited to the platform. Transitions between situations occur at the arrival or departure of a computing device. Adaptation is based on redistribution of UI components at the workspace level. In turn, this redistribution triggers an intra-modal GUI remolding at the dialogue controller level: when the control panel resides on the PDA, the note-editing task is no longer available. Adaptation is automatic: the user has no control over the adaptation process, but a minimum of meta-UI exists (i.e. the weaving effect) to express the transition between two situations. State recovery is performed at the physical action level: the slides show is not disturbed by adaptation.

4.2 The Sedan-Bouillon Website

Sedan-Bouillon is a website that aims to promote tourism in the regions of Sedan (France) and Bouillon (Belgium) (http://www.bouillon-sedan.com/). It provides tourists with information for visiting and sojourning in these regions including a selection of hotels, camping, and restaurants. Figure 52.5a shows a simplified version of this website when a user is logged in from a PC workstation.



a) b)

Figure 52.5 The Sedan-Bouillon web site. (a) UI centralized on a PC screen. (b) The control panel of the meta-UI to distribute UI workspaces across the resources of the interactive space. The lines of the matrix correspond to the workspaces, and the columns denote the browsers currently used by the same user.

Preparing a trip for vacation is an exciting experience when shared by a group of people. However, one single PC screen does not necessarily favor collaborative exploration. By dynamically logging to the same website with a PDA, users are informed on the PDA that they can distribute the UI components of the site across the interaction resources currently available. In the example of Fig. 52.5b, the user asks for the following configuration: the title must appear on the PDA as well as on the PC (the title slots are ticked for the two browsers available), whereas the content should stay on the PC and the navigation bar should migrate to the PDA. Figure 52.6 shows the resulting UI. At any time, the user can ask for a reconfiguration of the UI by selecting the "meta-UI" link in the navigation bar. The UI will be reconfigured accordingly.

Within the problem space of UI plasticity, the Sedan-Bouillon website is very similar to CamNote: same model of context of use, adaptation based on redistribution at the workspace level, with GUI intra-modal remolding at the workspace level. Contrary to CamNote, remolding is performed at the logical-presentation level (no task is suppressed or restructured), and state recovery is supported at the task level: if adaptation occurs as the user is filling a form, the content of the form is lost by the adaptation process. Contrary to CamNote, the user has full control over the reconfiguration of the UI using the control panel provided by the meta-

UI.

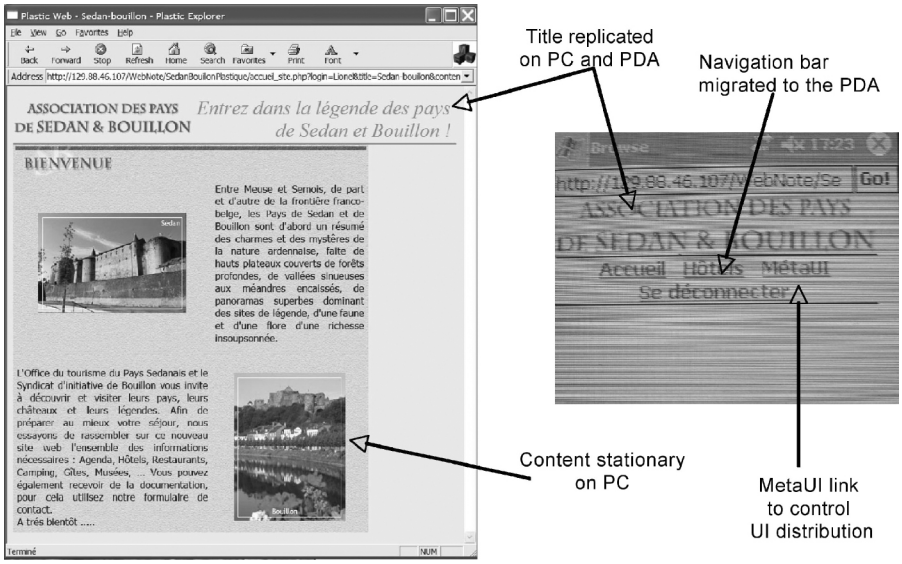


Figure 52.6 The Sedan-Bouillon web site when distributed across the resources of the interactive space. The MetaUI link allows users to return to the configuration panel shown in Figure 52.5b.

4.3 Photo-Browser

Photo-Browser supports photo browsing in a centralized or distributing way depending on the availability of a dynamic set of heterogeneous devices. These include a Diamond Touch interactive table, a wall, and a smart phone running Windows, MacOS X, and Android. The user interface of Photo-Browser is dynamically composed of:

- a Tcl-Tk component running on the multi-point interactive surface (Fig. 52.7-d),
- a Java component that shows a list of the image names (Fig. 52.7-b),
- an HTML-based browser to navigate through the images set (Fig. 52.7-c),
- as well as a Java component running on the gPhone to navigate sequentially through the photos using Next and Previous buttons (Fig. 52.8).

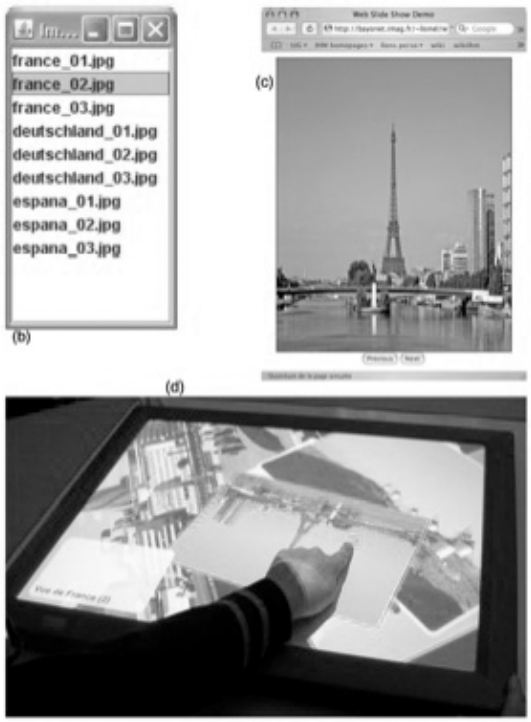


Figure 52.7 The Photo-browser application: a dynamic composition of executable and transformable components, managed by a dynamic set of interconnected factories running on different platforms (Windows, MacOS X, and Android).

The gPhone is dynamically connected to the interactive space by laying it down on the interactive table (Fig. 52.8, left). As part of the platform, the gPhone can be used as a remote-controller to browse photos displayed by the HTML UI component of Figure 52.7-c and video-projected on the wall.

Figure 52.8 (Left) Connecting a Gphone to the interactive space by laying it down on the interactive table. (Right) Using the Gphone as a remote-controller to browse photos displayed by the HTML UI component of fig. 52.7c and video-projected on the wall.

Within the problem space of UI plasticity, the context of use covered by Photo-Browser is a dynamic heterogeneous platform, adaptation is multi-TS based on redistribution at the interactor level (i.e. photos) with no remolding. In its current implementation, the meta-UI is simulated using the Wizard of Oz technique. This meta-UI includes the recognition of three gestures: a "wipe" gesture that allows the user to command the migration of the current selected photo from the table to the wall, the "wipe" gesture that commands the system to shut down the table and the contact of the gPhone with the Diamond Touch.

Having characterized three exemplars in the problem space of UI plasticity, we now consider the method and mechanisms necessary to support UI plasticity. Although we advocate a model-driven engineering (MDE) approach, we will analyze its limitations and suggest improvements.

5. MODEL-DRIVEN ENGINEERING

The motivation for MDE is the integration of knowledge and techniques developed in software engineering using the notions of model, model transformation, and mapping as the key concepts. In the early days of computer science, software systems were simple programs written in assembly languages. In those days, a code-centric approach to software development was good enough, not to say unavoidable, to ensure a fine control over the use of computing resources. Over the years, the field has evolved into the development of distinct paradigms and application domains leading to the emergence of multiple technological spaces (TS). Today, technological spaces can no longer evolve in autarky. Most of them share challenges of increasing complexity, such as adaptation, to which they can only offer partial solutions. Thus, we are in a situation where concepts, approaches, skills, and solutions need to be combined to address common problems. This is where MDE comes into play. MDE aims at achieving integration by defining gateways between technological spaces using a model-based approach. The hypothesis is that models, meta-models, model transformations, and mappings are everything.

5.1 Models

A *model is a representation of a thing* (e.g., a system), *with a specific purpose*. It is "able to answer specific questions in place of the actual" thing under study (Bézivin, 2004). Thus, a model, built to address one specific aspect of a problem, is by definition a simplification of the actual thing under study. For example, a task model is a simplified representation of some human activities (the actual thing under study), but it provides answers about how "representative users" proceed to reach specific goals.

A model may be *physical* (a tangible entity in the real world), *abstract* (an entity in the human mind), or *digital* (an entity within computers) (Favre, 2004a, 2004b). As illustrated in Fig. 52.9, a printed photograph of a young man named Peter is a physical representation of Peter that his mother (for example) uses for a specific purpose. Peter's mother has mental representations of him as a good son, or as a brilliant researcher (multiple abstract models about Peter). The authentification system that runs on Peter's computer knows him as a login name and password (digital model). If Peter's portrait is digitized as a JPEG picture, then the JPEG file is a digital model of a physical model. When displayed on the screen, the JPEG file is transformed into yet another digital graphics model in the system's main memory before being projected on the screen as an image (yet another physical model that Peter's mother can observe). As this example shows, models form oriented graphs (μ graphs) whose edges denote the m relation "is represented by". In other words, a model can represent another model, and a model can be represented by several models .

Models may be *contemplative* (not able to be processed automatically by computers) or *productive* (able to be processed by computers). Typically, scenarios developed in HCI (Rosson & Carroll, 2002) are contemplative models of human experience in a specified setting. In order to be processed (by humans, and/or by computers), a model must comply with some shared syntactic and semantic conventions: it must be a well-formed expression of a language. This is true both for productive and con-

templative models; most contemplative models developed in HCI use a mix of drawings and natural language. A language is the set of all well-formed expressions that comply with a grammar (along with semantics). In turn, a grammar is a model from which one can produce well-formed expressions (or models). Because a grammar is a model of a set of models, it is called a "meta-model."

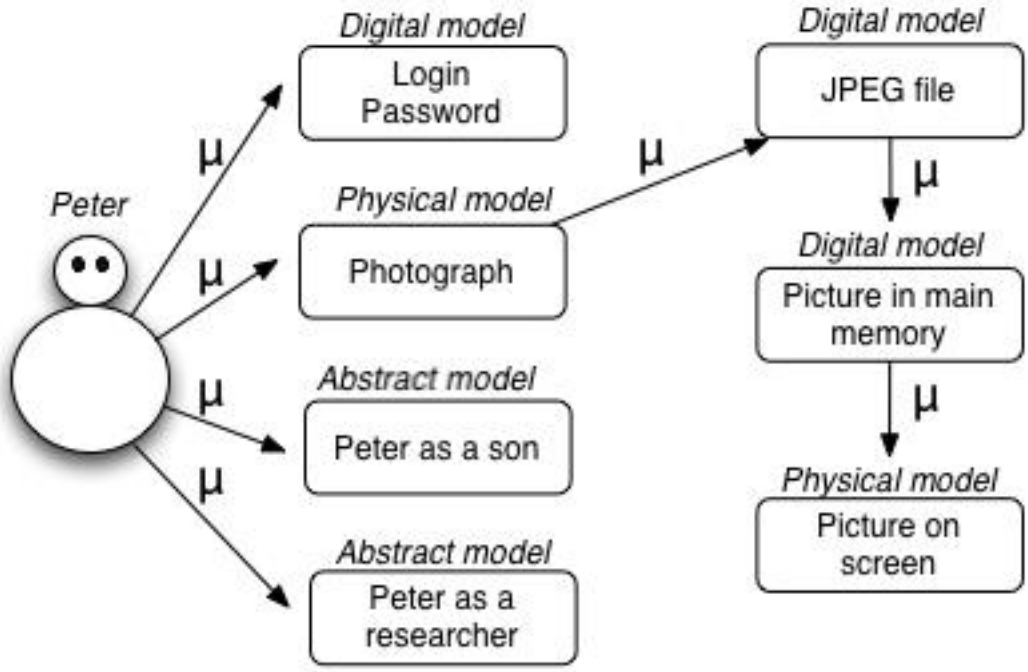


Figure 52.9 Models organized as oriented μ graphs.

5.2 Meta-model

A *meta-model is a model of a set of models that comply with it*. It sets the rules for producing models. It does not represent models. Models and meta-models form a tree: a model complies to a single meta-model, whereas a meta-model may have multiple compliant models.

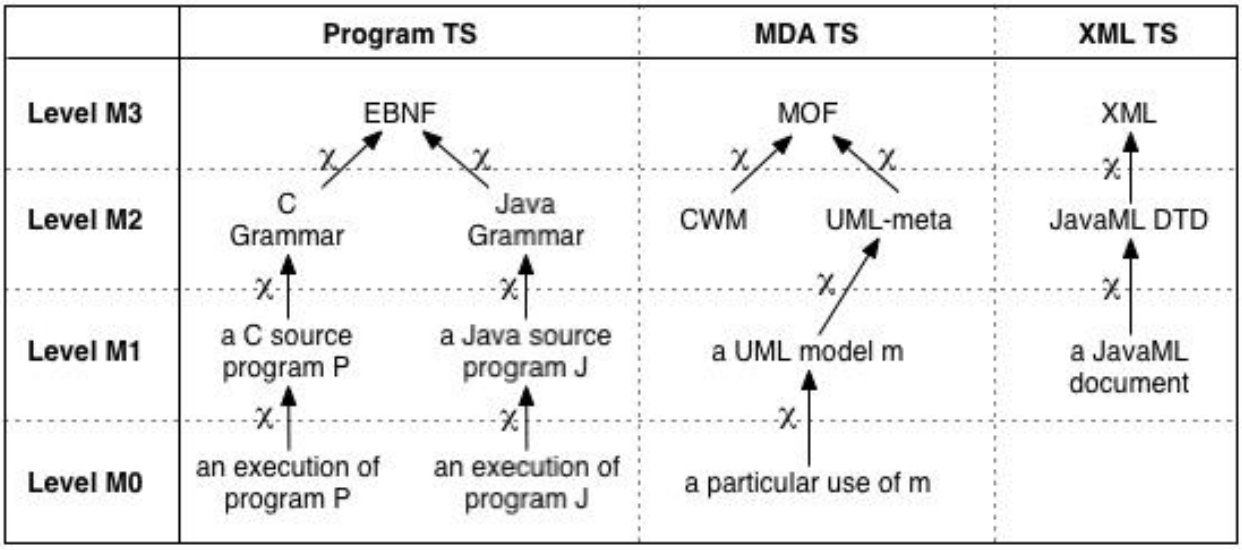


Figure 52.10 The OMG model-driven architecture four-layer stack.

As an example, suppose that the authentification system mentioned above is a Java program J. J is a digital model that represents Peter and that complies with the Java grammar G_J. G_J does not represent J, but defines the compliance of J with Java. G_J is one possible meta-model, but not the only one. The authentification system could also be implemented in C (yet another digital model of Peter). It would then be compliant with the C grammar G_C. Grammars G_C and G_J could, in turn, be produced from the same grammar such as EBNF (Extended Backus-Naur Form). EBNF, defined as the ISO/IEC 14977:1996 standard, is an example

of a meta-meta-model, which is a model of a set of meta-models that are compliant with it. It does not represent meta-models, but sets the rules for producing distinct meta-models. As shown in Fig. 52.10, the OMG model-driven architecture (MDA) initiative has introduced a four-layer modeling stack as a way to express the integration of a large diversity of standards using MOF (meta-object facility) as the unique meta-meta-model. MDA is a specific MDE deployment effort around industrial standards including MOF, UML, CWM, QVT, etc. EBNF, G_J and G_C, the Java and C programs are models that belong to the programming technological space. Within the MDA technological space, the java source code of our authentification system becomes a UML Java model compliant with the UML meta-model. In the XML Technological Space, the Java source code could be represented as a JavaML document compliant with a JavaML DTD (document type definition). (In the XML technological space, a DTD defines the legal building blocks of an XML document.)

As shown in Fig. 52.10, the relation ("complies with") makes explicit the multiplicity of existing technological spaces as well as their systematic structure into three levels of modeling spaces (the so-called M1, M2, M3 levels of MDA) plus the M0 level that corresponds to a system, or parts of a system. The μ and χ relations, however, do not tell how models are produced within a technological space nor how they relate to each other across distinct technological spaces. The notions of transformation and mapping are the MDE answer to this issue.

5.3 Transformations and Mappings

In the context of MDE, *a transformation is the production of a set of target models from a set of source models, according to a transformation definition*. A *transformation definition is a set of transformation rules* that together describe how source models are transformed into target models (Mens, Czarnecki & VanGorp, 2005). Source and target models are related by the τ relation "is transformed into." Note that a set of transformation rules is a model (a transformation model) that complies with a transformation meta-model.

Relation τ expresses an overall dependency between source and target models. However, experience shows that finer grain of correspondence needs to be expressed. Typically, the incremental modification of one source element should be propagated easily into the corresponding target element(s) and vice versa. The need for traceability between source and target models is expressed as mappings between source and target elements of these models. For example, as demonstrated in the next section, the correspondence between a source task (and concepts) and its target workspace, window and widgets, is maintained as a mapping function.

Transformations can be characterized within a four-dimension space:

- The *transformation may be automated* (it can be performed by a computer autonomously), it may be semi-automated (requiring some human intervention), or it may be manually performed by a human. For example, given our current level of knowledge, the transformation of a "value-centered model" into a "usability model" can only be performed manually. On the other hand, UI generators such as CTTE (Mori et al., 2002, 2004) produce user interfaces automatically from a task model.
- A *transformation is vertical* when the source and target models reside at different levels of abstraction. UI generation is a vertical top down transformation from high-level descriptions (such as a task model) to code generation. Reverse engineering is also a vertical transformation but it proceeds bottom up, typically from executable code to some high-level representation by the way of abstraction. A *transformation is horizontal* when the source and target models reside at the same level of abstraction. For example, translating a Java source code into C code preserves the original level of abstraction.
- Transformations are *endogenous* when the source and target models are expressed in the same language. They are *exogenous* when sources and targets are expressed in different languages while belonging to the same technological space. For example, the transformation of a Java source code program into a C program is exogenous (cf. Fig. 52.10).
- When *crossing technological spaces* (e.g., transforming a Java source code into a JavaML document), then additional tools (exporters or importers) are needed to bridge the gap between the spaces. Inter-technological transformations are key to knowledge and technical integration. This is the quest of MDE.

In the following section, we show how the MDE principles have been applied to the development of plastic interactive systems by bringing together HCI practice with mainstream software engineering.

6. CONTRIBUTIONS OF MDE TO HCI AND TO PLASTIC USER INTERFACES

The HCI community has a long experience with models and meta-models, long before MDE existed as a field. In the 1980's, grammars (meta-models) were the formal basis for generating textual and graphical user interfaces (Hayes et al., 1985; Shulert et al., 1985). MDE has helped the HCI community to define a shared vocabulary that express different perspectives on interactive systems as well as a reference framework for structuring the development process of plastic user interfaces.

6.1 Meta-models as different perspectives on an interactive system

Figure 52.11 shows an example of M2 level models that illustrate the principles of MDE applied to UI plasticity. These meta-

models (and their relations) are intended to specify the canonic structures of the "important" concepts of the problem space of UI plasticity. These include, but are not limited to the following:

- M2-Tasks and M2-Concepts, respectively define the notions of task and domain-dependent concepts. For example, in Fig. 52.11, a task has a name and pre- and post- conditions. It may be composed of subtasks by the way of a binary operator (such as the AND, OR, SEQ operators), or decorated with a unary operator (such as Optionality, Criticity, and Default option).
- M2-Abstract UI (AUI) is a canonical expression of the rendering and manipulation of the domain-dependent concepts in a way that is independent from the concrete interactors (widgets) available on the target platform. It is expressed in terms of workspaces (as in Mara (Sottet et al., 2006)), or in terms of Presentation Units (as in SEGUIA (Vanderdonckt et al., 1993; Vanderdonckt et al., 1999)), or in terms of Presentations (as in TERESA (Paternò, 1999)). Workspaces, Presentation Units and Presentations are synonyms to denote the same requirement: platform independence and absence of detailed UI design decisions.
- M2-Concrete UI (CUI) is an interactor-dependent expression of the user interface. An interactor (e.g., a widget provided by an interaction toolkit such as Swing) is an entity of the UI that users can perceive (e.g., text, image, animation) and/or manipulate (e.g., a push button, a list box, a check box). A CUI expresses detailed UI design decisions.
- M2-Final UI (FUI) is the effective UI as perceived and manipulated by the user at runtime. Typically, the same CUI java code may behave differently depending on the Java virtual machine used at runtime.
- M2-Context of use is defined as a specialization of the ontology presented in "Context and Context of Use" where users, platforms and physical environments are first class entities.
- M2-Transformation supports the description of transformations that can be automated. Figure 52.12 shows an example of M1-level transformation using ATL as a meta-model to express the transformation of M2-Task compliant models into M2-Workspace compliant abstract user interfaces. Figure 52.13 illustrates the principles of this transformation graphically.

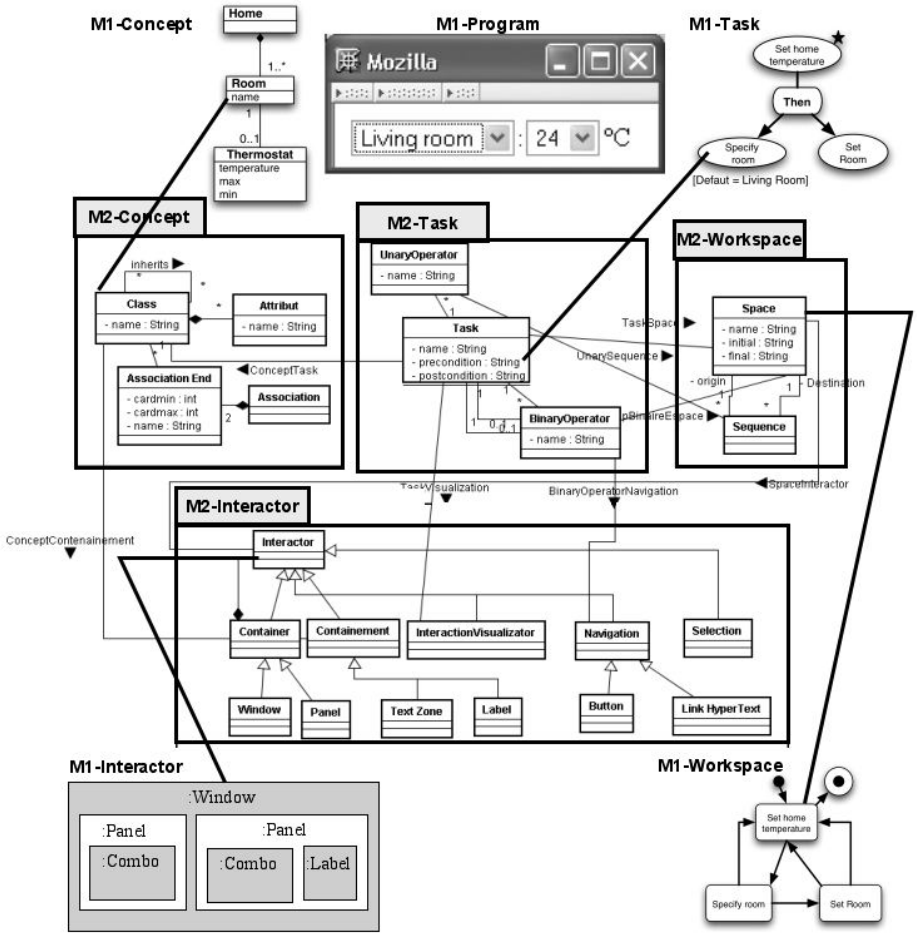


Figure 52.11 A home heating control system from an MDE perspective. A subset of the M1-level models and their mapping with their respective M2-level models. (For simplification purpose, only a subset of the mappings are represented.) [Adapted from (Sottet 20080].

Considering the relations between the M2 level models of Figure 52.11, the BinaryOperators of M2-Task are related to navigation interactors to move between workspaces in the Concrete UI. A task manipulates concepts (denoted by the "ConceptTask" relation). In turn, a concept is a class composed of a set of attributes. This class may inherit from other classes, and may serve different purposes. A concept is represented as interactor(s) in the Concrete UI by the way of the ConceptCon-

tainment relation. The TaskSpace relation shows how a task relates to a workspace. M2-Workspace structures the task space at an abstract level, and M2-interactor describes the interactors that will populate workspaces in the Concrete UI. As shown by the definition of M2-Workspace, workspaces are chained with each other depending on the binary operator of the source tasks. A workspace is mapped into the Concrete UI as a container class interactor. This container interactor, which, in the GUI modality, may be a panel or a window, is populated with interactors that render concepts and binaryOperations (navigation operators).

```
module M2TaskToM2Workspace {
  from M1Task : M2Task
  to   M1Workspace : M2Workspace
  -- One workspace per task
  rule TaskToSpace {
    from t : M2Task!Task
    to w : M2Workspace!Space (
      name <- t.name )
  }
  -- OrOperator to SequenceOperators
  rule OrOperatorToSequence{
    from o : M2Task!BinaryOperator (
      o.name = "or"
    )
    to motherToLeft : M2Workspace!Sequence (
      origin<- [ TaskToSpace.w] o.motherTask,
      destination<-[ TaskToSpace.w] o.leftTask)
```

Figure 52.12 An ATL transformation description based on the meta-models shown in Fig. 52.11. The rule *TaskToSpace* creates one workspace *w* per source task *t* where *w* takes the name of *t*; The rule *OrOperatorToSequence* transforms all OR operators *o* between two tasks (*o.leftTask* and *o.rightTask*) into two sequence operators (from o.*motherTask* to *o.leftTask*, and *o.leftTask* to *o.rightTask*).

In addition to examples of M2 level models, Figure 52.11 shows the M1 level models instantiated for a simple example: a home heating control system. At the top of Fig. 52.11, M1-task (which is compliant with M2-Task) has a name ("Set home temperature"), is repetitive (denoted by *), and is composed of two subtasks ("Specify room" and "Set Room") that must be executed in sequence. By default, if "Specify room" is not executed, then the selected room is the living room. M1-Workspace (at the bottom right of Fig. 52.11) is the Abstract UI that corresponds to M1-Task. M1-Workspace is comprised of three workspaces (one per task) whose relations (denoted by arrows) express the navigation scheme between the workspaces. M1-Workspace is then populated with interactors to obtain a Concrete UI. As shown at the bottom left of Fig. 52.11, the "Set home temperature" workspace is mapped into a window and the two others, "Specify room" and "Set Room" become panels populated respectively with a Combobox, and a Combobox with a Label. At the top center of Fig. 52.11, M1-Program represents the Final UI of the home heating control system. All of these M1 level models have been derived automatically by the way of transformations.

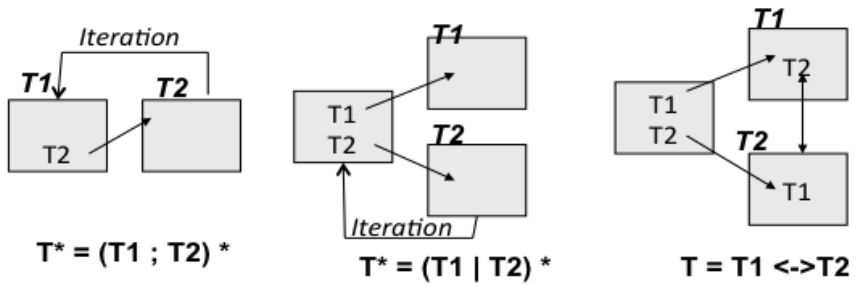


Figure 52.13 Typical transformation patterns between task models (expressed with UAN operators) and workspaces.

Until recently, transformation rules were implemented as code within UI generators offering very little to no control over the resulting user interface (Hayes et al., 1985; Schulert et al., 1985). In addition, mappings were limited to the expression of correspondence (bindings) between elements of the user interface with the API of the functional core (i.e. the business code). *MDE has helped the HCI community to promote transformation rules as models.* "Transformations as models" has three notable advantages – which, so far, has not been fully exploited by the HCI community: (1) It opens the way to knowledge capitalization and reuse: frequent transformations can serve as patterns in libraries, which in turn, provide handles for intra- and inter- UI consistency. (2) Comparative evaluations of UI's can be performed in a controlled way, and UI's can be (re)targeted for different contexts of use using different transformations. (3) Most notably, transformations can be transformed, offering a powerful formal recursive mechanism for supporting UI plasticity. To our best knowledge, no research has been conducted on transforming transformations for UI plasticity. On the other hand, patterns are emerging (Taleb et al., 2009) and early work has been initiated on user interfaces generated with different sets of transformation rules to support different usability criteria (Gajos et al. 2008; Sottet et al., 2007).

The set of meta-models presented in Figure 52.11 is only one example among a plethora of proposals. Whereas there are few meta-models for task modeling (e.g., TeresaXML), the CUI level, on the other hand, is a very active area of research with no clear integrated vision. This is primarily due to the inherent diversity of interaction modalities and of interaction paradigms developed in HCI. It is also the result of the diversity of technological spaces and of the competition among software vendors. To name just a few, the W3C recommendations include VoiceXML for voice integration, InkML for representing digital ink input with electronic pen, Extensible Multimodal Annotation Markup Language (EMMA) for multimodal input, not to mention 3D Markup Language (3DML) and Virtual Human Markup Language (VHML). Software vendors offer Macromedia Flex Markup Language (MXML), OpenLaszlo, XUL, XAML, and many other user interface description languages. As an answer to this plethora, the European ITEA2 UsiXML research project aims at covering the problem space of user interface plasticity into a unified, systematic and structured way with a clear motivation for standardization (http://usixml.org, http://itea.defimedia.be/usixml). The framework presented next will serve as a basis for structuring the development process along with the appropriate tool support.

6.2 A reference framework for a structured development process

Over the years, the CAMELEON framework has progressively come to serve as a generic canonical structure for exploiting MDE to address the problem of UI plasticity. In particular, the notion of transformation is used to cover many forms of development process (see Figure 52.14). Typically in a forward engineering process, an abstract UI is derived from the domain-dependent concepts and task models. In turn, the AUI is transformed into a concrete UI, followed by the final executable UI. At the opposite, a reverse engineering process infers abstract models from more concrete ones using vertical bottom-up transformations. Translations may also be applied to transform a source model into a new model adapted for a different context of use.

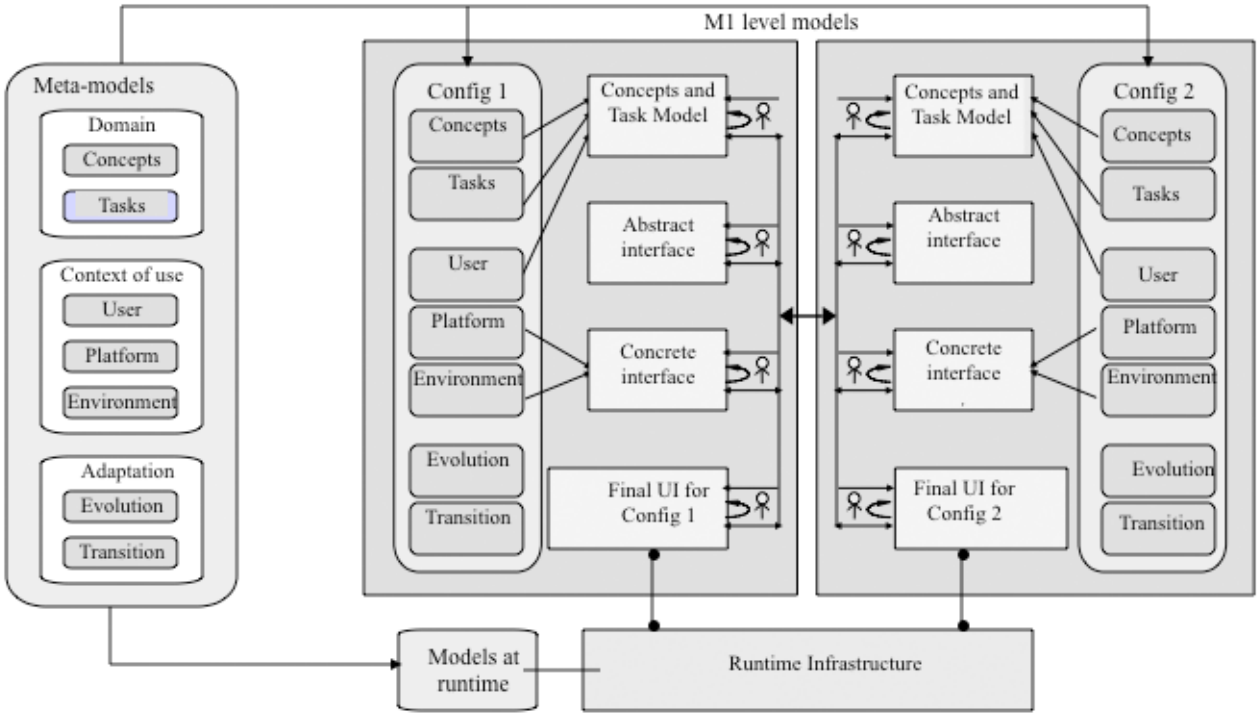


Figure 52.14 The CAMELEON reference framework for the development of plastic user interfaces.

Unlike the process initiated in the 1980's, which contained one entry point only at a high level of abstraction, the CAMELEON framework authorizes entry points at any level of abstraction from which any combination of horizontal and vertical

bottom-up and top-down transformations can be applied. This theoretical flexibility means that the stakeholders involved in the development of an interactive system can use the development process that best suits their practice or the case at hand. In other words, the CAMELEON framework can be put in actions in many ways.

Seminal work in forward engineering for UI plasticity includes UIML (Abrams, Phanariou, Batongbacal, Williams, & Shuster, 1994) and XIML (Puerta & Eisenstein, 2001) that transform M1 level models into M0 level programs to support Logical Presentation level adaptation for centralized GUI. Tools for retargeting UI's such as Vaquita (Bouillon & Vanderdonckt, 2002) and WebRevenge (Paganelli & Paternò, 2003) correspond to a combination of bottom-up vertical, horizontal, and top-down vertical transformations. They lie within the same meta-meta level (the XML Technological Space), but they use distinct M2 meta-models. Vaquita and WebRevenge work off line. On the other hand, Digymes (Coninx, Luyten, Vandervelpen, Van den Bergh, & Creemers, 2003) and Icrafter (Ponnekanti, Lee, Fox, Hanrahan, & Wilograd, 2001) generate Concrete User Interfaces (CUI) at runtime where a renderer dynamically computes a CUI from a workspace level model expressed in XML. Websplitter (Han, Perret, & Naghshineh, 2000) supports the distribution of web pages content at the interactor level across the interaction resources of heterogeneous clusters, but distribution is statically specified in an XML policy file. As proof of concepts, small size exemplars have been developed for different technological spaces.

To summarize, the CAMELEON reference framework is an MDE-compliant conceptual generic structuring conceptual tool for the development of plastic UI's:

- As a structuring reference framework, it federates the HCI community around a consensus.
- As a conceptual generic tool, it sets a vast agenda for technical research.
- As an MDE-compliant framework, it is still unclear in practice that formal modeling is the only way to go in HCI. This issue is discussed next.

6.3 The limits of MDE

The CAMELEON reference framework brings together the "right models" but the HCI community is far from having the "models right". As discussed above, the profusion of initiatives related to user interface description languages (UIDL) is symptomatic of the need – and difficulty, to define a coherent set of non-ambiguous and easy to understand meta-models capable of covering the problem space of plastic UI's. In our opinion, two meta-models (at least) are key to the success of MDE for addressing UI plasticity: transformations and Concrete UI's.

As stated above, transformations offer an elegant mechanism for full flexibility and technical integration. However, *transformations are hard to express*: QVT and ATL (Bézivin et al., 2003) are not languages for naïve developers. In addition, usability rules (Sottet et al., 2007) are even harder to convey formally. More importantly, *inverse transformations cannot be automatically derived* for any source transformations. This is a fundamental flaw that may result in inconsistent models as transformations are performed up and down iteratively during the life cycle of a system, breaking down the flexibility of the solution space envisioned by the CAMELEON reference framework. TransformiXML of the UsiXML (Limbourg 2004; Limbourg, Vanderdonckt, Michotte, Bouillon, & Lopez-Jacquero, 2004) meta-level environment, which is based on graphs transformations, is certainly a promising way.

At the CUI level, meta-modeling, not only lags behind innovation, but bridles creativity. UIDL's for the expression of concrete user interfaces are technology-driven instead of leaving rooms for new forms of interaction techniques. Although the CARE properties (Coutaz et al., 1995) have been devised 15 years ago, CUI languages have hardly scratched the surface of multimodal interaction. We are still unable to generate the paradigmatic "put-that-there" multimodal user interface introduced more than 25 years ago (Bolt, 1980). We do however generate simplistic multimodal UI's based on XHTML+VoiceXML but with very limited micro-dialogues for interaction repair (Berti et al., 2005). Actually, CUI-level UIDL's are still struggling with the description of conventional GUI's for desktop computing. Meanwhile:

- New forms of "constructable" computers such as the MIT shiftables[2] and the CMU toy blocks[3] are put on the market;
- Novel interaction techniques are proliferating whether it be for supporting mobility (e.g., SixthSense (Mistry et al., 2009)), for 3D interaction (where gesture and 3D screens are becoming predominant), or even for graphical tabletops and multi-surface interaction (Balakrishnan et al., 2009);
- New requirements are emerging: design is switching from the development of useful and usable systems for people with precise goals, to engaging and inspired interaction spaces whose users can easily switch from consumers to creators.

In short, CUI meta-models need to capture the unbound vibrant convergence of physicality with "digitality". Perhaps, meta-modeling is, by essence, the wrong approach to CUI's: a model, which represents a thing, is necessarily a simplification, there-

[2] http://sifteo.com/

[3] http://www.modrobotics.com/

fore a reduction, of the real thing. In these conditions, the subtle aspects of interaction, which make all the differences between constrained and inspired design, are better expressed using code directly in place of an abstraction of this code. However this assertion should be mitigated by the following findings: designers excel at sketching pictures to specify concrete rendering. On the other hand, they find it difficult to express the dynamics, forcing them to use natural language (Myers et al., 2008). One way to fill the gap between designers' practice and productive models is to revive work à-la-Peridot (Myers, 1990) such as SketchiXML (Coyette et al., 2004; Kieffer et al., 2010) where drawings are retro-engineered into machine-computable rendering. As for inferring behavior from examples, the promising "Watch What I Do" paradigm initiated in the late 1970's (cf. Dave Smith's Pygmalion system (Smith, 1993)) is still an opened question.

In addition to impeding creativity, *Model Driven Engineering, as a software development methodology, has favored the dichotomy between the design stages and the runtime phase*, resulting in three major drawbacks:

- Over time, models may get out of sync with the running code.
- Design tools are intended for software professionals, not for "the people". As a result, end-users are doomed to consume what software designers have decided to be good for their hypothetic target users.
- Runtime adaptation is limited to the changes of context identified as key by the developers. Again, the envelope for end-users' activities is constrained by design.

Applied to UI development, the dichotomy between design and runtime phases means that UI generation from a task model cannot cope with ambient computing where task arrangement may be highly opportunistic and unpredictable. On the other hand, because the task model is not available at runtime, the links between the FUI and its original task model are lost. It is then difficult, not to say impossible, to articulate runtime adaptation based on semantically rich design-time descriptions. As a result, a FUI cannot be remolded beyond its cosmetic surface as supported by the CSS.

Blurring the distinction between the design stage and the runtime phase is a promising approach. This idea is emerging in main stream middleware (Ferry et al., 2009) as well as in HCI. The middleware community, however, does not necessarily address end-user concerns. Typically, a "sloppy" dynamic reconfiguration at the middleware level is good enough if it preserves system autonomy. It is not "observable" to the end-user whereas UI re-molding and UI redistribution are! Thus, UI plasticity puts additional constraints on the developers and on the tools to support them. In particular, it becomes necessary to make explicit the transition between the source and the target UI's so that, in Norman's terms, end-users can evaluate the new state. We need to pay attention to transition UI's in generic terms, not on a case per case basis.

In the following section, we show how Model Driven Engineering can be reconciled with a code-centric approach at runtime.

7. MODELS AT RUNTIME

The combination of MDE with "code-centricity" relies on three principles:

- Principle #1: Cooperation between closed-adaptiveness and open-adaptiveness,
- Principle #2: Runtime availability of high-level models,
- Principle #3: Balance between the importance of Principles #1 and #2. The application of this principle is illustrated with Photo-Browser.

7.1 Principles #1: Cooperation between closed-adaptiveness and open-adaptiveness

"A system is open-adaptive if new application behaviors and adaptation plans can be introduced during runtime. A system is closed-adaptive if it is self-contained and not able to support the addition of new behaviors" (Oreisy et al., 1999). As discussed above, by design, an interactive system has an "innate domain of plasticity": it is closed-adaptive for the set of contexts of use for which this system/component can adapt on its own. For unplanned contexts of use, the system is forced to go beyond its domain of plasticity. It must be open-adaptive so that a tier infrastructure (i.e. a middleware) can take over the adaptation process. The CAMELEON Runtime conceptual architecture (CAMELEON-RT) shows how closed-adaptiveness and open-adaptiveness can be combined harmoniously (Balme et al., 2004). CAMELEON–RT shown in Figure 52.15 is a refinement of the box "Runtime infrastructure" at the bottom of Figure 52.14.

At the bottom of Figure 52.15, "Hardware" denotes a wide variety of physical entities: computing and communication facilities, interaction resources such as displays, mice, and stylus, as well as sensors and actuators. "Operating Systems" includes legacy OS such as Linux, MacOS and Android, virtual machines such as the JVM, and modality interpreters such as speech and gesture recognition. Together, "Hardware" and "Operating Systems" constitute the ground-basis of the interactive space, i.e. the platform as defined in Section "User Interface Plasticity: Definition".

The top of Figure 52.15 shows the interactive systems (e.g., CamNote and Photo-Browser) that users are currently running in the interactive space. The Meta-UI is one of them. A flower-like shape, ⬡, denotes open-adaptive components of these interactive systems. Components are open-adaptative if they provide the world with management mechanisms. Management mechanisms include self-descriptive meta-data (such as the current state and the services it supports and requires), and the methods to control its behavior such as start/stop and get/set-state. Software reflexivity coupled with a component model is

a good approach to achieve open-adaptiveness. The miniature adaptation-manager shape, [illegible], denotes facilities embedded in the interactive system to support closed-adaptiveness to observe the world, to detect situations that require adaptation, to compute a reaction that satisfies the new situation, and to perform adaptation. This functional decomposition is similar to that of the tier infrastructure shown in the center of Figure 52.15.

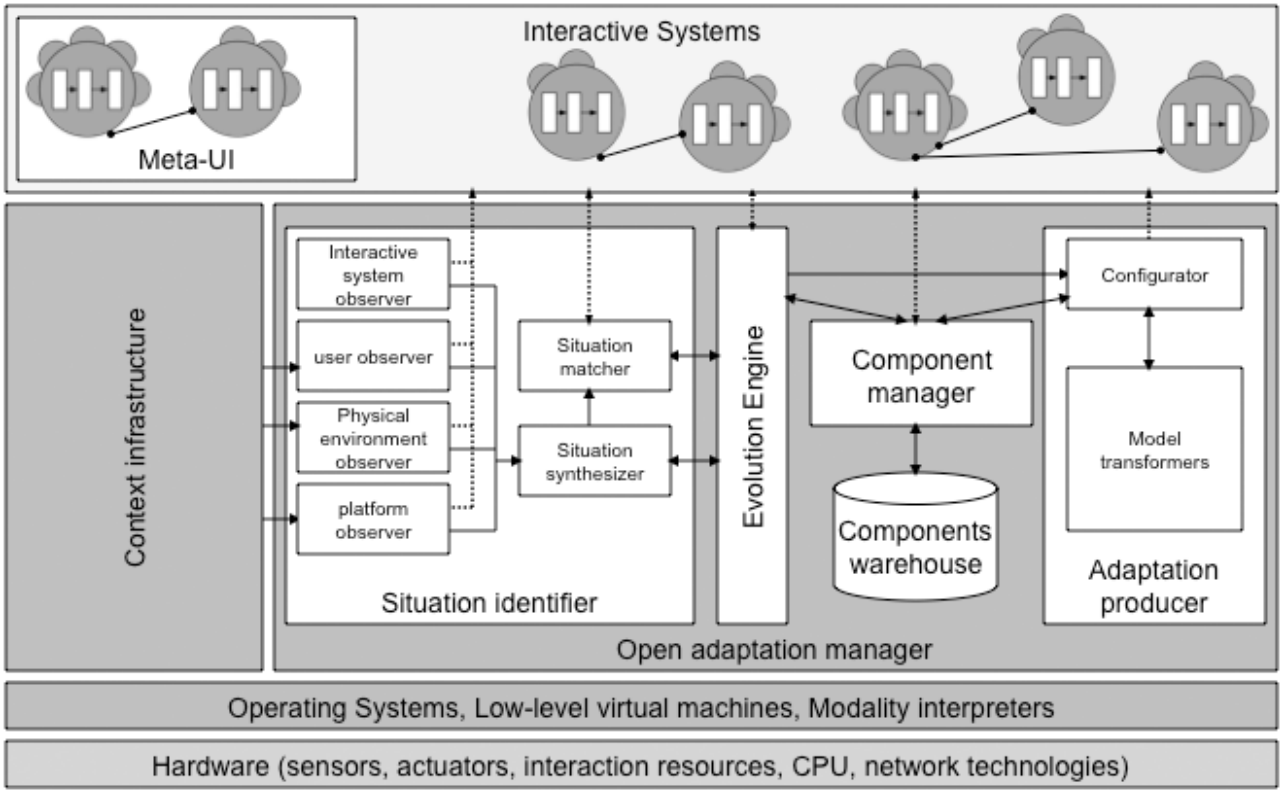


Figure 52.15 CAMELEON RT: a functional decomposition for supporting a mix of closed-adaptiveness and open-adaptiveness at runtime.

The tier infrastructure that supports open-adaptiveness is structured in the following way:

- The context infrastructure builds and maintains a model of the context of use (Reignier et al., 2007). In turn, this infrastructure can be refined into multiple levels of abstraction, typically: raw data acquisition as numeric observables, transformation of raw data at the appropriate level of abstraction (e.g., as symbolic observables) which then feeds into situation management.
- The situation synthesizer computes the situation and possibly informs the evolution engine of the occurrence of a new situation. (This layer is in general considered as part of the context infrastructure.)
- The evolution engine elaborates a reaction in response to the new situation.
- The adaptation producer implements the adaptation plan produced by the evolution engine. This is where the following dimensions of the problem space of UI plasticity come in play: granularity of UI remolding and/or redistribution, granularity of state recovery, coverage of technological spaces, and presence of a meta-UI.

Such a functional decomposition is commonly used for the development of autonomic systems. To adapt this decomposition to plastic UI's, we propose the following improvements:

- The end-user is kept in the loop: the reaction to a new situation may be a mix of specifications provided by developers or learnt by the evolution engine based on observations of, and reasoning on human and environmental behavior. In addition, the evolution engine as well as the adaptation producer may call upon end-users' advice by the way of the meta-UI.
- The components referred to in the action plan do not necessarily exist as executable code. This is where Principle #2 comes into play.

7.2 Principle #2: Runtime availability of high-level of abstraction models

At runtime, an interactive system is a set of graphs of models that express different aspects of the system at multiple levels of abstraction. As advocated by the CAMELEON framework, these models are related by mappings and transformations. As a result, an interactive system is not limited to a set of linked pieces of code. Models developed at design-time, which convey high-level design decision, are still available at runtime for performing rational adaptation beyond cosmetic changes. When a component retrieved by the component manager is a high-level description such as a task model, the configurator relies on reificators to produce executable code as in Digymes (Coninx et al., 2003) and iCrafter (Ponnekanti et al., 2001). A retrieved component may be executable, but may not fit the requirements. Ideally, it can be reversed-engineered through abstractors,

then transformed by translators and reified again into executable code (Bouillon et al., 2002).

7.3 Principle #3: Balance between Principles #1 and #2

By analogy with the slinky meta-model of the Arch model (Bass et al., 1992), the software developer can play with principles #1 and #2. At one extreme, the interactive system may exist as one single task model linked to one single AUI graph, linked to a single CUI graph, etc. (see Figure 52.16). This application of Principle #1 does not indeed leave much flexibility to cope with unpredictable situations unless it relies completely on the tier middleware infrastructure that can modify any of these models on the fly, then triggers the appropriate transformations to update the Final UI. This approach works well for interactive systems for which conventional WIMP user interfaces are "good enough".

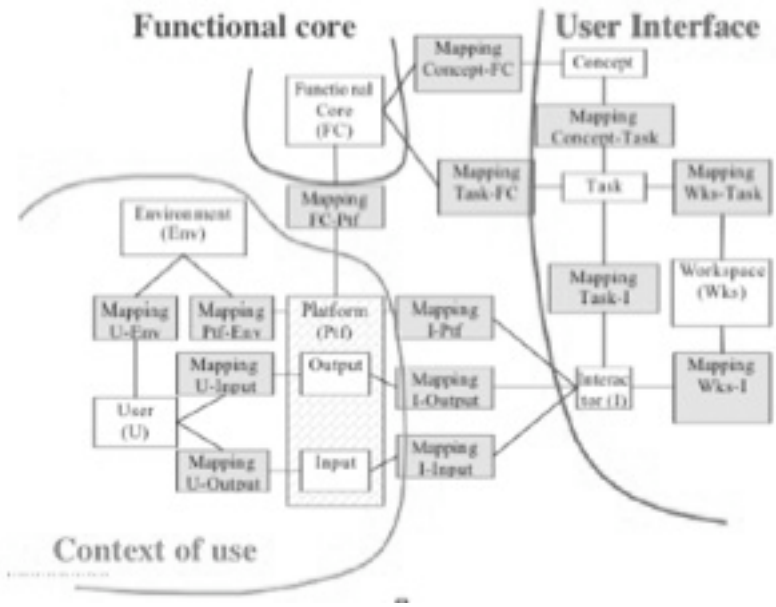


Figure 52.16 An interactive system as a graph of models available at runtime. These models are related by mappings and transformations.

At the other extreme, the various perspectives of the system (task models, AUI, FUI, context model, etc.) as well as the adaptation mechanisms of the tier infrastructure are distributed across distinct UI service-oriented components, each one covering a small task grain that can be run in different contexts of use. This approach has been applied in the Comet toolkit (Demeure et al., 2008).

Basically, a Comet is a plastic micro-interactive system whose architecture pushes forward the separation of concerns advocated by PAC (Coutaz, 1987) and MVC (Krasner et al., 1988). The functional coverage of a comet is left open (from a plastic widget such as a control panel, to a complete system such as a powerpoint-like slide viewer). Each Comet embeds its own task model, its own adaptation algorithm, as well as multiple CUI's and FUI's, each one adapted to a particular context of use. FUI's are hand-coded possibly using different toolkits to satisfy our requirements for fine-grained personalization and heterogeneity. From the infrastructure point of view, a Comet is a service that can be discovered, deployed and integrated dynamically into the configuration that constitutes an interactive environment. The COTS (Bourguin et al., 2007), whose executable UI code is meta-described with the task they support, are based on similar ideas.

Figures 52.7 and 52.8 show another application of principles #1 and #2 for the implementation of Photo-browser. The FUI of Photo-browser is dynamically composed of:

- a Tcl-Tk component running on a multi-point interactive surface (Fig. 52.7-d),
- a Java component that shows a list of the image names (Fig. 52.7-b),
- and an HTML-based browser to navigate through the images set (Fig. 52.7-c).

Photo-browser is implemented on top of a tier middleware infrastructure (called Ethylene) that covers the evolution engine, the component manager as well as the adaptation producer of Figure 52.15. Ethylene is a distributed system composed of Ethylene factories each one running on possibly different processors (IP devices). The role of an Ethylene factory is to manage the life cycle of a set of components that reside on the same IP device as this factory, and that have been registered to this factory. When residing on storage space, a component is meta-described using EthylenXML, an extension of the W3C standard WSDL (Web Service Definition Language). This meta-description includes the human task that the component supports, the resources it requires, and whether it is executable code or transformable code. In the latter case, it may be a task model, an AUI, a CUI, or even a graph of these models. For example, the HTML-based component (Fig 52.7.c) is a CUI expressed in a variation of HTML. It must be transformed on the fly to be interpreted by an HTML renderer. The Tcl-Tk multi-point UI and the Java list are executable code. Their EthyleneXML meta-description specifies that they support image browsing and image selection tasks, that they need such and such interaction resources (e.g., a Tcl-Tk interpreter and a Diamond Touch

interactive table) for proper execution, and that they require such and such communication protocol to be interconnected with other components. The Gphone UI component is an executable Gphone app that supports the next-previous browsing tasks (fig. 57-8). Interconnection between the components is initiated by the factories.

As shown by the examples above as well as by other work (Blumendorf et al. 2010; Clerckx et al. 2007; Duarte et al. 2006; Savidis et al. 2010), the engineering community of HCI has focused its attention on runtime adaptation of the UI portion of an interactive system, not on the dynamic adaptation of the interactive system as a whole. The software engineering community is developing several approaches to enable dynamic bindings for service-oriented architectures. For example, Canfora et al. propose the dynamic composition of web services based on BPEL4People (that expresses a task-like model) as well as an extension of WSDL to meta-describe the services and using these two descriptions to generate the corresponding user interface (Canfora et al., 2009). Although bindings can be performed at runtime, users are confined within the workflow designed by the software developers. In addition, the generated UI's are limited to conventional WIMP user interfaces.

One promising approach to support flexibility at runtime, is to consider the functional core components as well as UI components as services. In Ethylene, UI components adhere to this philosophy. They can be implemented in very different technologies, they can be discovered and recruited on the fly based on their meta-description, and they can be transformed on the fly. On the other hand, the business logic side of interactive systems is left opened. CRUISe (Pietschmann et al., 2009) aims at supporting both sides in a uniform way, but applies to the dynamic composition of web services and UI composition for the web (Yu et al., 2007).

8. CONCLUSION

Model-Driven Engineering has provided the HCI community with useful concepts for framing its own research agenda. Additional research is required for the definition of meta-models, transformations and mappings provided that high-level descriptions can take full advantage of the latest innovations at the FUI level. Models at design time should not disappear at runtime, but should be available to go beyond cosmetic adaptation. Design phase and runtime phase equal "même combat!"
Maximum flexibility and quality should be attainable by modeling the business logic as well as the user interface as services with their own domain of plasticity. UI components should not be pure executable code. They have to be meta-described to express their exact nature and contracts with a human-centered perspective. They can be retrieved, transformed, and recomposed on the fly thanks to a tier middleware infrastructure. This middleware, which supports context, dynamic discovery as well as the dynamic (re)composition of business logic and of transformable UI components, will permit interactive systems to go beyond their domain of plasticity. However, we must be careful at keeping the user in the loop while being able to produce transition user interfaces automatically.

The risk is that this wonderful apparatus will be designed for the specialists. We need to put the power in the people's hands and explore the potential from social programming. The success of the Apple App Store is a good indication for this. Mash-up tools have also started this trend for composing web-based applications (e.g., Google Gadgets or Yahoo! Widgets). More collaboration should be developed with the "cloud computing crowd". After all, an interactive space is a mini-cloud. If interaction resources were virtualized as memory, network and computing resources are currently envisioned by the "systemers", then this would simplify enormously the development of user interfaces.

In short, MDE is an important tool for adaptation as long as it does not block creativity.

9. ACKNOWLEDGMENTS

This work is the result of research supported by the CAMELEON European project (IST-2000-28323) and by the Framework VI Network of Excellence SIMILAR. We thank the ANR CONTINUUM project (ANR-08-VERS-005) as well as the ITEA 08026 UsiXML project for supporting this research. The authors wish to thank Gaëtan Rey for the development of the contextor infrastructure, Lionel Balme and Alexandre Demeure for the implementation of CamNote and the Sedan-Bouillon website as well as for the development of the first version of the runtime infrastructure for plastic UI's. Thanks to Jean-Marie Favre and Jean-Sébastien Sottet for their insights into MDE principles.

References

Abrams, M., Phanariou, C., Batongbacal, A., Williams, S., & Shuster, J. (1994). UIML: an appliance-independent XML User Interface Language. Proceedings of the 8th WWW Conference, WWW'94.

Balakrishnan, R. & Baudisch, P. (2009). Special Issue on Ubiquitous Multi-Display Environments, Human-Computer Interaction, 2009, Vol. 24, Taylor and Francis.

Balme, L., Demeure, A., Barralon, N., Coutaz, J., & Calvary, G. (2004). CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces, Lecture Notes in Computer Science, Vol. 3295, 2004. In P. Markopoulos, B. Eggen, E. Aarts et al. (Eds), Ambient intelligence: Second European Symposium, EUSAI 2004. Springer-Verlag Heidelberg (Publisher), ISBN: 3-540-23721-6, Eindhoven, The Netherlands, November 8–11, 291–302.

Bass, L., Faneuf, R., Little, R., Mayer, N., Pellegrino, B., Reed, S., Seacord, R., Sheppard, S., & Szczur, M. (1992). Arch, a meta-model for the runtime architecture of an interactive system. The UIMS Developers Workshop. SIGCHI Bulletin, 24(1), ACM Publ., 32–37.

Berti, S., & Paternò F. (2005). Migratory multimodal interfaces in multidevice environments. In Proceedings International Conference on Multimodal Interfaces (ICMI 05), ACM Publ., 92–99.

Bézivin, J., Dupé, G., Jouault, F., Pitette, G., & Rougui, J. (2003). First experiments with the ATL transformation language: transforming XSLT into Xquery. OOPSLA Workshop, Anaheim California USA.

Bézivin, J. (2004). In search of a basic principle for model driven engineering. Novatica Journal, Special Issue, March–April 2004.

Blumendorf, M., Leehmann, G & Albayrak, S. (2010). Bridging models and systems at runtime to build adaptive user interfaces. In Proc. of the 2010 ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2010, ACM Publ., 2010, 9-18.

Bolt, R. (1980). Put That There": Voice and gesture at the graphics interface. In Proc. of the 7th International Conf. on Computer Graphics and Interactive techniques, ACM Publ., 1980, 262-270.

Bouillon, L., & Vanderdonckt, J. (2002). Retargeting web pages to other computing platforms. Proceedings of IEEE 9th Working Conference on Reverse Engineering WCRE'2002 (Richmond, 29 October–1 November 2002), IEEE Computer Society Press, Los Alamitos, 339–348.

Bourguin, G., Lewandowski , A. &Tarby, J.-C. (2007). Defining Task Oriented Component. In Proc. TAMODIA 2007, Lecture Notes in Computer Science 4849 Springer 2007, ISBN 978-3-540-77221-7, 2007, 170-183.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., & Vanderdonckt, J. (2003). A Unifying Reference Framework for Multi-target User Interfaces. Interacting with Computers, Elsevier Science B.V., June, 2003, 15(3), 289-308.

Canfora, G., Di Penta, M., Lombardi, P. & Villani, M.L. (2009). Dynamic Composition of Web Applications in Human centered Processes. IEEE PESOS'09, May 18-19, 2009.

Clerckx, T., Vandervelpen, C. & Coninx, K. (2007). Task-based design and runtime support for multimodal user interface distribution. In Proc. Of Engineering Interactive Systems, 2007.

Cockton, G. (2004). From quality in use to value in the world. In ACM Proceedings CHI 2004, Late Breaking Results, 1287–1290.

Cockton, G. (2005). A development framework for value-centred design. In ACM Proceedings CHI 2005. Late Breaking Results, 1292–1295.

Coninx, K., Luyten, K., Vandervelpen, C., Van den Bergh, J., & Creemers, B. (2003). Dygimes: Dynamically generating interfaces for mobile computing devices and embedded Systems. In Proceedings Mobile HCI, 256–270.

Coutaz, J. (1987). PAC, An Implemention Model for Dialog Design. In Proceedings of Interact'87, Stuttgart, September, 1987, 431-436.

Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J. & Young, R. (1995). Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE properties, Proceedings of the INTERACT'95, Chapman&Hall Publ., 1995, 115-120.

Coutaz, J., Crowley, J., Dobson, S., & Garlan, D. (2005). Context is key. Communications of the ACM, ACM Publ., 48(3), 49–53.

Coutaz, J. (2006). Meta User Interfaces for Ambient Spaces. In Proc. TAMODIA 2006, 5th International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2006, 2006, Springer Verlag publ.

Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q. & Vanderdonckt, J. (2004) SketchiXML: towards a multi-agent design tool for sketching user interfaces based on USIXML. In Proceedings of the 3rd Annual Conference on Task Models and Diagrams, TAMODIA 2004, Prague, Czech Republic, 2004.

Demeure, A., Calvary, G., Koninx, K. (2008). A Software Architecture Style and an Interactors Toolkit for Plastic User Interfaces. In Proceeding of the 15th international workshop DSV-IS 2008, LNCS, Springer Berlin, 2008, 225-237.

Dey, A. K. (2001) Understanding and using Context. Journal of Personal and Ubiquitous Computing, Springer London, Vol 5, 2001, 4-7.

Dourish, P. (2001) Where the Action Is: The Foundation of Embodied Interaction. MIT Press, Cambridge, 2001.

Duarte, C. & Carriço, L. (2006). A conceptual framework for developing adaptive multimodal applications. In Proc. of the 11th international conference on Intelligent User Interfaces, IUI'06, ACM, New York, 2006, 132-139.

Elrad, T., Filman, R., & Bader, A. (2001). Aspect oriented programming. Special issue, Communication of the ACM, 44(10), 28–95.

Favre, J. M. (2004a). Foundations of model (driven) (reverse) engineering. Dagsthul Seminar on Language Engineering for Model Driven Development, DROPS. http://drops.dagstuhl.de/portals/04101.

Favre, J. M. (2004b). Foundations of the meta-pyramids: Languages and meta-models. DROPS, http://drops.dagstuhl.de/portals/04101.

Ferry, N. Hourdin, G., Lavirotte, S., Rey, G., Tigli, J.-Y. & Riveill, M. (2009). Models at Runtime: Service for Device Composition and Adaptation. In 4th International Workshop Models@run.time, Models 2009 (MRT'09), 2009.

Gajos, K., Wobbrock, J. & Weld, D. (2008). Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces. In CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, New York, NY, USA, ACM, 2008, 1257-1266.

Gaver, W., Bowers, J., Boucher, A., Pennington, S. &Villar, N. (2006). The History tablecloth: Illuminating Domestic Activity. In Proceedings of the 6th conference on Designing Interactive Systems, ACM, 2006, 199-208.

Han, R., Perret, V., & Naghshineh, M. (2000). WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing. ACM Conference on Computer Supported Cooperative Work (CSCW 2000), 221–230.

Harrison, C., Desney, T. & Morris, D. (2010). Skinput: Appropriating the Body as an Input Surface. In Proceedings of CHI'10, the 28th international conference on human factors in computing systems, ACM, 2010, 453-462.

Hartson, R., Siochi, A., & Hix, D. (1990). The UAN: a user-oriented representation for direct manipulation interface designs. ACM Transaction on Information Systems (TOIS), 8(3), 181–203.

Hayes, P .J., Szekely, P., & Lerner, R. A. (1985). Design alternatives for user interface management systems based on experience with COUSIN. In Proceedings of the ACM Conference on Human Factors in Computing Systems CHI'85. San Francisco, CA, Apr. 14–18, 169–175.

Kieffer, S., Coyette, A. & Vanderdonckt, J. (2010). User Interface Design by Sketching: A complexity Analysis of Widget Representations. In Proc. Of the 2010 ACM SIGCHI Symposium on Engineering Interactive Computing Systems, ACM Pub., 2010, 57-66.

Krasner, G.E. & Pope, S.T. (1988). A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. Journal of Object Oriented Programming (JOOP), 1(3), 1988, 26-49.

Kurtev, I., Bézivin, J., & Aksit, M. (2002). Technological spaces: An initial appraisal. CoopIS, DOA'2002 Federated Conferences. Industrial Track, Irvine.

Limbourg, Q. (2004). Multi-path development of user interfaces. PhD of University of Louvain La Neuve, Belgium.

Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., & Lopez-Jaquero, V. (2004). UsiXML: a Language Supporting Multi-Path Development of User Interfaces. Proceedings of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems, EHCI-DSVIS'2004 (Hamburg, July 11–13, 2004).

Mens, T., Czarnecki, K. & Van Gorp, P. (2005). A taxonomy or model transformations. Dagstuhl Seminar Proceedings 04101. http://drops.dagstuhl.de/opus/volltexte/2005/11.

Merrill, D., Kalanithi, J., and Maes, P. (2007). Siftables: Towards Sensor Network User Interfaces. In Proceedings of the 1st international

Conference on Tangible and Embedded interaction, 2007.
Mistry, P. & Maes, P. (2009). SixthSense – A Wearable Gestural Interface. In Proc. SIGGRAPH Asia 2009, Emerging Technologies, Yokohama, Japan, 2009.
Mori, G., Paternò, F., & Santoro, C. (2002). CTTE: Support for developing and analyzing task models for interactive system design. IEEE Transactions on Software Engineering, August 2002, 797–813.
Mori, G., Paternò, F., & Santoro, C. (2004). Design and development of multidevice user interfaces through multiple logical descriptions. IEEE Transactions on Software Engineering, August 2004.
Myers, B. (1990). Creating User Interfaces using programming by example, visual programming, and constraints. ACM Transaction on Programming Languages and Systems (TOPLAS), Vol. 12 (2) (1990), ACM Publ., 143-177.
Myers, B. (2001). Using handhelds and PCs together. Communication of the ACM, 44(11), 34–41.
Myers, B., Park, S.Y., Nakano, Y., Mueller, G. & Ko, A. (2008). How designers Design and Program Interactive Behaviors. in Proc. IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC), 2008, 177-184.
Nielsen, J. (1993). Usability engineering. London Academic Press. ISBN 0-12-518406-9.
Oreizy, P., Gorlick, M., Taylor, R., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D. & Wolf, A. (1999). An Architecture-Based Approach to Self-Adaptive Software", IEEE Intelligent Systems, May/June, 1999, 54-62.
Paganelli, L., & Paternò, F. (2003). A tool for creating design models from website code. International Journal of Software Engineering and Knowledge Engineering, World Scientific Publishing, 13(2), 169–189.
Paternò, F. (1999). Model-based Design and Evaluation of Interactive Applications, Springer Verlag, Berlin, 1999.
Paternò, F. (2003). Concur Task Trees: An engineered notation for task models. In D. Diaper, & N. Stanton, (Eds.), The Handbook of Task Analysis for Human-Computer Interaction (pp. 483–503 ch. 24). Lawrence Erlbaum Associates.
Pietschmann, S., Voigt, M. & MeiBner, K. (2009). Dynamic Composition of Service-Oriented Web User Interfaces. Proc. of the 4th International Conf. on Internet and Web Applications and Services, ICIW 2009, IEEE CPS, ISBN 9780769536132, 2009.
Ponnekanti, S., Lee, B., Fox, A., Hanrahan, P., & Winograd, T. (2001). Icrafter: A service framework for ubiquitous computing environments. In G. Abowd, B. Brumitt, S. Shafer (Eds.) Proceedings Ubicomp 2001, Springer Publ., LNCS 2201, 57–75.
Puerta, A. & Eisenstein, J. (2001). XIML: A common representation for interaction data. Proceedings IUI01, ACM publ., 214–215.
Reignier, P., Brdiczka, O., Vaufreydaz, D., Crolwey, J.L. & Maisonnasse, J. Context Aware Environments : from Specification to Implementation. Expert Systems: The Journal of knowledge Engineering, Vol 5/24, 2007, 304-320
Rekimoto, J. (1997). Pick and drop: A direct manipulation technique for multiple computer environments. In Proceedings. of UIST97, ACM Press, 31–39.
Reignier, P. Brdiczka, O., Vaufreydaz, D., Crowley, J.L. & Maisonnasse, J. (2007). Contexte-Aware Environments: from Specification to Implementation. Expert Systems: The Journal of Knowledge Engineering, 2007.
Rey, G. (2005). Le Contexte en interaction homme-machine: le contexteur. PhD Thesis, Université Joseph Fourier, France.
Rosson, M. B., & Carroll, J. (2002). Usability Engineering. Scenario-based development of human computer interaction. Morgan Kaufmann.
Savidis, A. & Stephanidis, C. (2010). Software refactoring process for adaptive user interface composition. In Proc. of the 2010 ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2010, ACM Publ., 2010, 19-28.
Shackel, B. (1984). The concept of usability. In J. Bennett et al. (eds), Visual display terminals: Usability issues and health concerns, Englewood Cliffs NJ: Prentice-Hall, ISBN 0-13-942482-2.
Schulert, A. J., Rogers, G. T., & Hamilton, J. A. (1985). ADM-A dialogue manager. In Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'85), San Francisco, CA, Apr. 14–18, 177–183.
Smith, D. C. (1993). Pygmalion: An executable Electronic Blackboard. Chapter1 In "Watch What I Do", A. Cypher ed., The MIT Press, 1993.
Sottet, J.-S., Calvary, G. & Favre, J.-M. (2006). Models at runtime for sustaining user interface plasticity. Int Models@run.time workshop, in conjunction with MODELS/UML 2006.
Sottet, J.-S., Calvary, G., Coutaz, J. &Favre, J.-M. (2007). A Model-Driven Engineering Approach for the Usability of User Interfaces. In Proc. Engineering Interactive Systems (EIS2007), J. Gulliksen et al. (eds), LNCS 4940, 2007, 140-157.
Stephanidis, C., & Savidis, A. (2001). Universal access in the information society: Methods, tools, and interaction technologies. Journal of the Universal Access in Information Society UAIS, 1(1), 40–55.
Streitz, N., Geibler, J., Holmer, T., Konomi, S., Müller-Tomfelde, C., Reischl, W., Rexroth, P., Seitz, P., & Steinmetz, R. (1999). i-LAND: An interactive Landscape for Creativity and Innovation. In Proceedings. of the ACM conference On Human Factors in Computer Human Interaction (CHI99), ACM, 120–127.
Taleb, M., Seffah, A. & Abran, A. (2009). Interactive Systems Engineering: A Pattern-Oriented and Model-Driven Architecture. In Software Engineering Research and Practice, 2009, 636-642.
Thevenin, D., & Coutaz, J. (1999). Plasticity of User Interfaces: Framework and Research Agenda. In A. Sasse & C. Johnson (Eds.) Proceedings Interact99, Edinburgh, IFIP IOS Press, 110–117.
Van Lamsweerde, A. (2009). Requirements Engineering: From System Goals to UML Models to Software Specifications, Wiley, 2009.
Vanderdonckt, J., & Bodard, F. (1993). Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. Proceedings of the joint ACM Conference on Human Factors in Computing Systems CHI and IFIP Conference on Human Computer Interaction INTERACT, April 24–29, 1993, Amsterdam, The Netherlands, ACM Press.
Vanderdonckt, J., & Berquin, P. (1999). Towards a Very Large Model-based Approach for User Interface Development. In Proc. of 1st Int. Workshop on User Interfaces to Data Intensive Systems UIDIS'99 (Edimburg, 5-6 September 1999), N.W. Paton & T. Griffiths (eds.), IEEE Computer Society Press, Los Alamitos, 1999, 76-85.
Winograd, T. (2001). Architectures for Context. Human-Computer Interaction – Special Issue on Context-Aware Computing, Lawrence Erlbaum Associates, 16 (2-4), 2001, 401-420.
Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F. M. & Matera, M.. A Framework for rapid Integration of Presentation Components. In WWW'07 Proc. of the 16th International Conf. on World Wide Web, 2007, 923-932.